VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IDENTIFIKACE OBLIČEJE NA PLATFORMĚ ANDROID

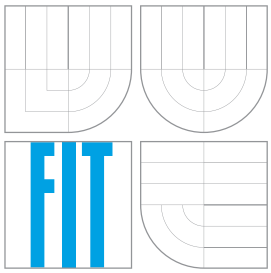
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN KARHÁNEK

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IDENTIFIKACE OBLIČEJE NA PLATFORMĚ ANDROID

FACE IDENTIFICATION ON ANDROID PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN KARHÁNEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ LÁNÍK

BRNO 2011

Abstrakt

Tato práce popisuje možnosti využití identifikace osob podle obličeje na mobilních zařízeních s platformou Android. Čtenáři je přiblížena struktura tohoto systému a způsob vytváření aplikací pro něj. Jsou zde uvedeny některé metody použitelné při identifikaci a ty z nich, použité v praktické části, jsou rozebrány podrobněji. Další součástí práce je popis implementace aktivního modelu vzhledu (AAM) v nativním kódu pro použití na mobilním zařízení a zhodnocení výsledků použitých algoritmů.

Abstract

This work describes ways to use a person identification based on faces on mobile devices with Android platform. A reader is introduced into a structure of this system and a way to create applications for it. Besides, there are also methods usable to the face identification. Some of these methods (used in an implementation) are described in more detail. This work also contains a description of model AAM (Active Appearance Model) for implementing in mobile devices and evaluation of used algorithms results.

Klíčová slova

Android, mobilní zařízení, smartphone, obličej, identifikace obličeje, rozpoznání obličeje, klasifikace, PCA, AAM

Keywords

Android, mobile device, smartphone, face, face identification, face recognition, classification, PCA, AAM

Citace

Martin Karhánek: Identifikace obličeje na platformě Android, diplomová práce, Brno, FIT VUT v Brně, 2011

Identifikace obličeje na platformě Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Aleše Láníka.

.....
Martin Karhánek
25.5.2011

Poděkování

Chtěl bych poděkovat Ing. Aleši Láníkovi za odborné vedení a konstruktivní připomínky, které mi při tvorbě této práce pomohly.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V několika posledních letech dochází k obrovskému pokroku ve vývoji mobilních zařízení. Tradiční mobilní telefony začínají být vytlačovány sofistikovanějšími zařízeními, tzv. *chytrými telefony* – smartphony. Tyto se stávají cenově dostupnějšími a rozšiřují se tak mezi širokou veřejnost. Na trhu je několik velkých hráčů, nabízejících různé operační systémy, více či méně otevřené uživatelům a vývojářům. Ačkoliv je toto odvětví poměrně mladé, postupuje vývoj mílovými kroky. Některé parametry a periferie jsou už považovány za součást standardní výbavy. Zařízení tak většinou disponují např. fotoaparátem (kamerou), který byl ale součástí mobilních telefonů už dříve. Další samozřejmostí je bezdrátové připojení pomocí WiFi či Bluetooth, polohové snímače a akcelerometry, nebo GPS, díky níž může telefon nahradit autonavigaci. Častou součástí výbavy bývá dotykový displej, díky němuž na jednu stranu narostla velikost zobrazovací plochy, na druhou stranu je použití klasické hardwarové klávesnice v mnoha ohledech daleko pohodlnější a poskytuje potřebnou zpětnou vazbu, kterou např. vibrace nahrazují jen omezeně. Proto bývá využití výsuvné hardwarové klávesnice častým plusem.

Dík všem těmto vlastnostem a vzrůstajícímu výkonu roste možnost využití v mnoha oblastech a vznikají aplikace jak pro denní potřebu, tak specializované, které nejsou ani tak zaměřené na obyčejného uživatele, jak na otestování možností, jež tato zařízení nabízí. Vedle nejrůznějších kalendářů, přehrávačů médií a her tak vznikají například projekty, testující algoritmy počítačového vidění a zpracování obrazu. Další oblastí zájmu jsou aplikace rozšířené reality, která tím, že zařízení jsou malá a mobilní, nabývá nových rozměrů.

Tato práce má za cíl otestovat možnosti využití mobilního zařízení pro identifikaci osob pomocí snímků obličeje. Druhá kapitola této práce popisuje strukturu systému Android a uvádí některé zásadní informace, důležité pro vývoj aplikací na něm. V následujících dvou kapitolách jsou rozebrány teoretické možnosti identifikace a podrobněji popsány některé vybrané algoritmy. V poslední části je potom popsána struktura implementované aplikace a zvolené metody – aktivního modelu vzhledu (AAM).

Platforma Android byla zvolena za cílovou kvůli její otevřenosti uživatelům i vývojářům. Přes své relativní mládí je tento systém čím dál více rozšířen a disponuje kvalitně zpracovanou dokumentací a jinými zdroji.

Praktické využití je možné vidět v automatické identifikaci a označování osob na fotkách uložených v mobilním telefonu. Zajímavou myšlenkou by bylo propojení se sociálními sítěmi.

Kapitola 2

Android

Android je softwarová platforma vyvinutá společností Google a určená pro mobilní zařízení (smartphony, netbooky, PDA, ...). Momentálně je její vývoj pod taktovkou sdružení firem OHA (Open Handset Alliance), jehož členem je mimo společnosti Google i několik dalších firem (celkem necelých 50), pohybujících se v oblasti vývoje a výroby výpočetní techniky, mobilních zařízení, někteří mobilní operátoři a další. Cílem je vytvoření platformy maximálně otevřené vývojářům i uživatelům.

Mimo operačního systému, za který je někdy označován, je jeho součástí middleware a některé základní aplikace. Android SDK pak nabízí nástroje a API potřebné pro vývoj aplikací pro tuto platformu v jazyce Java [1].

Součásti a vlastnosti systému

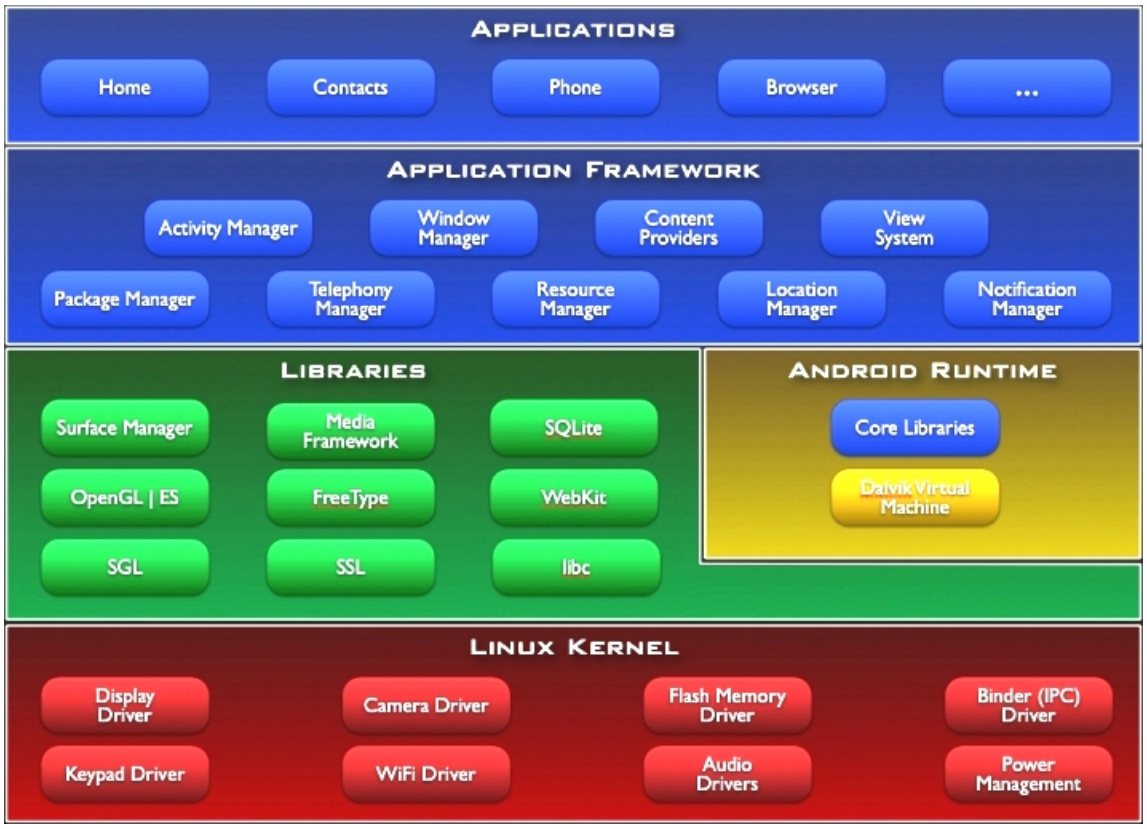
- **Aplikační framework** – umožňující správu, nahrazování a znovupoužití komponent.
- **Dalvik virtual machine** – registrově orientovaný virtuální stroj optimalizovaný pro mobilní zařízení.
- **Integrovaný webový prohlížeč** – postavený na opensource enginu WebKit[1].
- **Optimalizovaná grafika** – používá vlastní knihovnu pro 2D grafiku. 3D grafika je založena na specifikaci OpenGL ES 1.0 (u zařízení, která to umožňují, je možná hardwarová akcelerace, ale není nutná).
- **SQLite** – pro strukturované ukládání dat.
- **Podpora médií** – pro běžné audio a video formáty a obrazové formáty (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- **GSM telefonie**. *
- **Bluetooth, EDGE, 3G, WiFi**. *
- **Kamera, GPS, kompas, akcelerometr**. *
- **Bohaté vývojové prostředí** – emulátor, nástroje pro debuggování, paměťový a výkonový profiling a plugin pro vývojové prostředí Eclipse.

* Závislé na hardwaru konkrétního přístroje [2].

[1] http://webkit.org/

2.1 Architektura systému Android

Android je postaven na Linuxovém jádru, konkrétně verzi 2.6, které zajišťuje systémové funkce jako bezpečnost, správu paměti a procesů, síť a ovladače hardwarových komponent. Funguje jako abstraktní vrstva mezi hardwarem a softwarovou vrstvou. Struktura systému je zobrazena na 2.1 [2] .



Obrázek 2.1: Diagram ukazující hlavní komponenty operačního systému. Převzato z [1].

Aplikace

Mezi základními aplikacemi, které jsou součásti systému, je např. poštovní klient, aplikace pro správu a odesílání SMS, kalendář, mapy, webový prohlížeč, správa kontaktů a další.

Aplikační framework

Stejné rozhraní používané aplikacemi, které jsou přímo součástí systému, je přístupné vývojáři a ten je může použít k vývoji vlastních aplikací. Architektura je navržena tak, aby bylo snadné jednotlivé komponenty opakovaně využít [2].

Základním prvkem všech aplikací je sada služeb a systémů obsahující:

- **Views** – prvky grafického rozhraní (seznamy, tabulky, tlačítka, ...).
- **Content Providers** – umožňující aplikacím přístup k datům jiných aplikací, nebo sdílení jejich vlastních dat.

- **Resource Manager** – zprostředkovává přístup ke zdrojům, které nejsou přímo součástí zdrojového kódu (grafika, řetězce pro různé lokalizace programu, soubory s popisem vzhledu aplikace).
- **Notification Manager** – pro upozorňování uživatele na události např. zobrazením zprávy ve stavovém řádku, přehráním zvuku, blikáním LED diody, vibracemi.
- **Activity Manager** – správa životního cyklu aplikace.

Knihovny

Některé knihovny jejichž funkce jsou zprostředkovávány přes Aplikační framework [2].

- **System C library** – implementace standardní systémové knihovny C (libc) odvozená od BSD implementace a upravená pro vestavěná linuxová zařízení.
- **Media Libraries** – postavené na PacketVideo's OpenCORE, umožňující manipulaci s běžnými audio, video a obrazovými formáty (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG).
- **Surface Manager** – přístup k zobrazovacího subsystému.
- **SGL** – základ pro 2D grafiku.
- **3D libraries** – implementace založená na API OpenGL ES 1.0. Umožňuje využívat hardwarovou 3D akceleraci.
- **FreeType** – knihovna pro vykreslování vektorového a bitmapového písma.
- **SQLite** – pro použití relačních databází.

Android Runtime

Součástí je sada knihoven (*Core Libraries*), které poskytují většinu funkcionality základních knihoven jazyka Java.

Každá aplikace běží jako samostatný proces s vlastní instancí DVM (Dalvik Virtual Machine). DVM je vytvořen tak, aby byl možný efektivní běh několika virtuálních strojů současně. Před spuštěním je aplikace převedena do formátu `.dex` (Dalvik Executable), který je vhodný pro zařízení s omezenou pamětí a rychlostí procesoru. Od verze platformu 2.2 navíc DVM provádí kompilaci JIT (*Just In Time*), takže aplikace je překládána průběžně za běhu [2, 20].

2.2 Vývoj aplikací

Aplikace pro Android jsou psány v jazyku Java (v případě nativního kódu v jazyku C nebo C++). K distribuci aplikací pak slouží formát *Android package*, což je archiv s koncovkou `.apk`, který obsahuje zkompilovaný Java kód a ostatní potřebná data a soubory. Balíček má pevnou adresářovou strukturu a jeho součástí je také soubor `AndroidManifest.xml` [2].

2.2.1 Soubor `AndroidManifest.xml`

Tento soubor musí mít každá aplikace ve svém kořenovém adresáři. Obsahuje informace o aplikaci, které systém potřebuje znát před jejím spuštěním. V souboru manifest jsou mimojiné uvedeny následující informace [2]:

- Název Java balíčku, ve kterém je aplikace umístěna. Tento název slouží jako unikátní identifikátor aplikace.
- Popisuje komponenty aplikace – aktivity, služby, broadcast receivers, content providers – ze kterých je aplikace složena. Uvádí třídy, které implementují každou komponentu, a její možnosti.
- Uvádí, ke kterým chráněným částem API vyžaduje aplikace přístup a jaké povolení musí mít ostatní aplikace, aby mohly s touto komunikovat.
- Je zde uvedena minimální verze API.
- Seznam knihoven, které aplikace používá.

Příklad manifest souboru:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="test.helloworld"
      android:versionCode="1"
      android:versionName="1.0">
   <application android:icon="@drawable/icon"
         android:label="@string/app_name"
         android:permission="android.permission.CAMERA">
      <activity android:name=".MainActivity"
                android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
         </intent-filter>
      </activity>
   </application>
   <uses-sdk android:minSdkVersion="8" />
</manifest>
```

Jde o příklad jednoduché aplikace s jednou aktivitou a s povolením přístupu ke kameře (fotoaparátu). U některých prvků je vidět, že např. řetězce nejsou uváděny přímo, ale je použit odkaz s názvem. Tyto řetězce jsou pak uloženy v dalším *resource* souboru (`strings.xml`).

2.2.2 Komponenty aplikace

Jednou z hlavních vlastností platformy Android je, že jedna aplikace může používat prvky jiné aplikace (pokud k tomu má povolení). Přitom není třeba, aby kód těchto prvků byl součástí aplikace, nebo aby byly jinak připojeny. Pokud je to potřeba, je jednoduše spuštěna část cizí aplikace, která tento prvek poskytuje.

Systém ale musí být schopen spustit pouze část aplikace, která je potřeba, a vytvořit pro ni instance potřebných objektů. Z toho důvodu nemá aplikace jeden přístupový bod jak je obvyklé, ale je složená z komponent, které je možné spustit samostatně. Existují čtyři typy těchto komponent: Aktivita (Activity), Služba (Service), Broadcast receiver, Content provider [1, 2].

Aktivity (Activities)

Aktivita představuje jeden prvek uživatelského rozhraní, se kterým uživatel může v jeden moment pracovat. Např. u aplikace pro odesílání textových zpráv může být jednou aktivitou komponenta pro zobrazování seznamu kontaktů. Aplikace může obsahovat jednu, nebo více Aktivit. Obvykle jedna z aktivit pak bývá hlavní a bývá tedy vstupním bodem aplikace, pokud je tato spuštěna klasicky uživatelem.

Každá Aktivita má vlastní okno pro vykreslování obsahu, které většinou zaujímá velikost celého displeje. Může být ale i menší a pak bývá překreslované přes okna v pozadí. Obsah okna je tvořen hierarchií prvků odvozených od třídy `View`, což je základní stavební prvek GUI. Obsah je aktivitě přiřazován pomocí metody `Activity.setContentView()`, které je jako parametr přidán kořenový prvek hierarchie [1].

Služby (Services)

Služby nemají vizuální podobu, ale běží na pozadí. Může to být například proces, který komunikuje přes síť, nebo provádí nějaké výpočty a výsledky poskytuje některé aktivitě.

Je možné se připojit k již běžící službě nebo spustit službu, pokud zatím neběží. Po dobu připojení je možné se službou komunikovat přes její rozhraní [1].

Broadcast receivers

Jde o komponentu, která pouze přijímá a reaguje na broadcastové zprávy. Příkladem systémového broadcastu může být informace o nízkém stavu baterie, příchozí hovor nebo textová zpráva.

Broadcast receiver nemá grafické rozhraní, ale může jako reakci na nějakou zprávu spustit aktivitu [1].

Content providers

Content provider zpřístupňuje data aplikace ostatním aplikacím. Tato data mohou být uložena přímo v souborovém systému, v SQL databázi, nebo jinak. Základem je třída `ContentProvider`, která implementuje metody pro přístup k datům. Ostatní aplikace ale nevolají metody této třídy přímo, ale používají objekt `ContentResolver`, který s providerem komunikuje [1].

2.2.3 Intent

Intent je objekt určený pro posílání zpráv mezi komponentami. Je to pasivní datová struktura, nesoucí abstraktní popis operace, která má být provedena, nebo v případě broadcastu popis něčeho, co se stalo či má být oznámeno. Komponenty jsou asynchronně aktivovány zasíláním těchto zpráv. Výjimkou je komponenta Content Provider, která je aktivována, pokud je adresátem požadavku od Content Resolveru. Každá komponenta je aktivována jiným způsobem.

Aktivitě je možné při spuštění předat Intent jako parametr `Context.startActivity()` nebo `Activity.startActivityForResult()` pokud např. jedna Aktivita spouští jinou a očekává od ní nějaký výsledek. Výsledek pak předává volaná Aktivita Intentem, který je použit při volání `onActivityResult()`. Aktivita může získat přístup k Intentu, který jí byl při spuštění předán, metodou `getIntent()`. Pokud je metoda znovuspuštěna s jiným Intentem, je zavolána metoda `onNewIntent()`.

Předáním Intentu metodou `Context.startService()` je spuštěna služba, nebo jsou existující službě předány nové instrukce. Systém poté zavolá metodu `onStart()`, kterou předá službě Intent. Podobně může být použito `Context.bindService()` k vytvoření spojení mezi volající komponentou a cílovou službou. Služba obdrží Intent zavoláním její `onBind()` metody.

Vytvořit broadcast může aplikace zavoláním funkce `Context.sendOrderedBroadcast()`, `Context.sendBroadcast()`, nebo `Context.sendStickyBroadcast()`. Systém doručí Intent všem příjemcům broadcastu zavoláním jejich metody `onReceive()` [1, 2].

Intent filtr

Pokud Intent explicitně uvádí cílovou komponentu, Android ji najde a aktivuje. Pokud ale není cíl přímo uveden, systém musí vybrat nejvhodnější komponentu, která na něj odpoví. To provádí pomocí Intent filtrů, které jsou uvedeny v souboru `AndroidManifest.xml`. Každý filtr informuje o schopnosti komponenty zpracovat nějaký druh Intentu [1, 2].

2.2.4 Životní cyklus komponent

Každá komponenta má určitý životní cyklus, který začíná vytvořením její instance. V průběhu existence pak může přecházet do různých stavů.

Aktivita

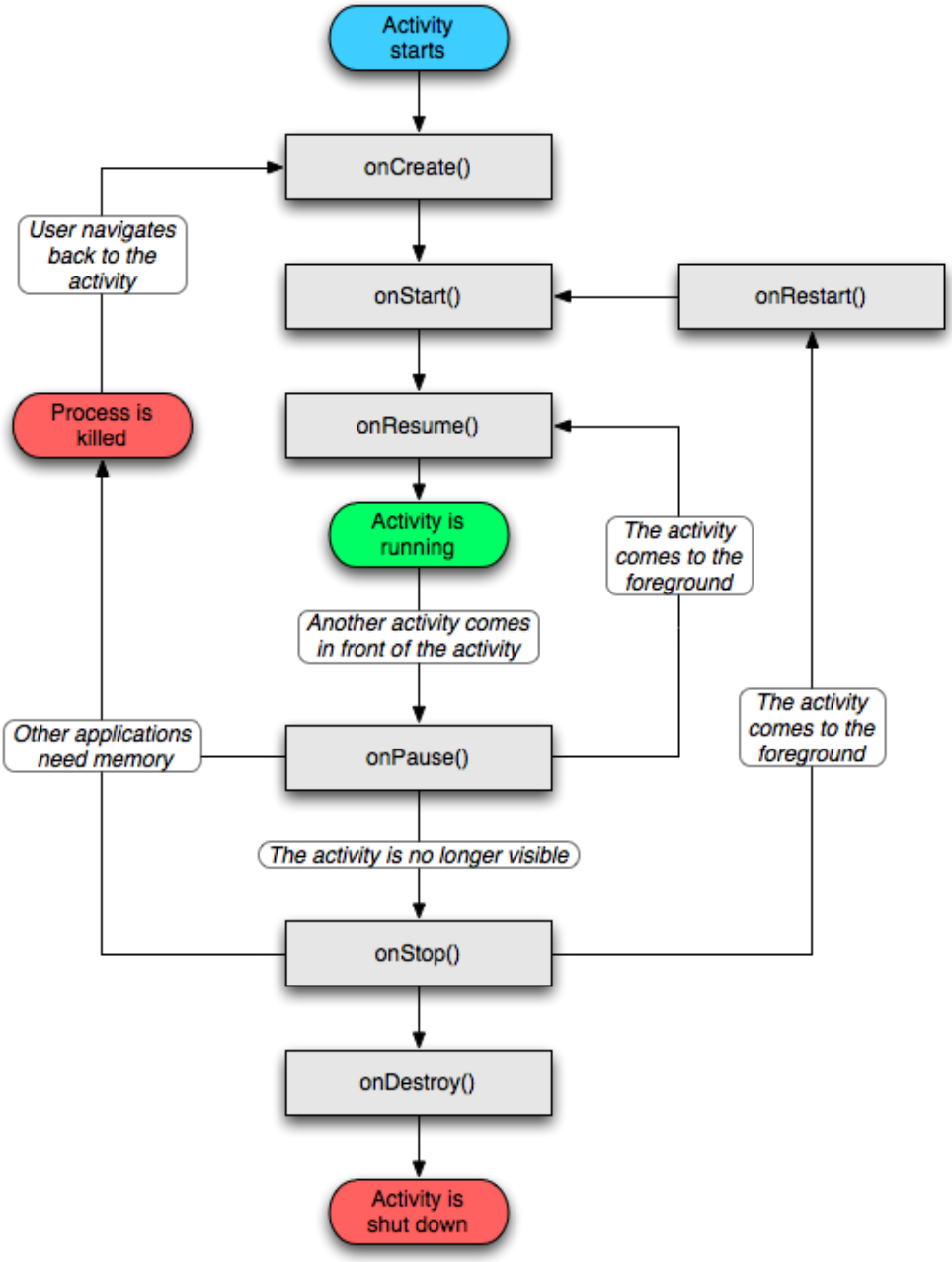
Aktivita se může nacházet v třech různých stavech:

- *Active* nebo *running* – tj. aktivní resp. běžící je Aktivita pokud je v popředí a uživatel s ní právě pracuje.
- *Paused* – Aktivita je stále viditelná, ale uživatel pracuje s jinou Aktivitou. V podstatě stále běží a udržuje si všechna svá data, ale může být ukončena v případě velmi nízkého stavu paměti.
- *Stopped* – Aktivita není viditelná, je úplně překryta jinou, se kterou je pracováno. Stále udržuje informace o svém stavu a datech, ale může být kdykoliv ukončena, pokud systém potřebuje uvolnit paměť.

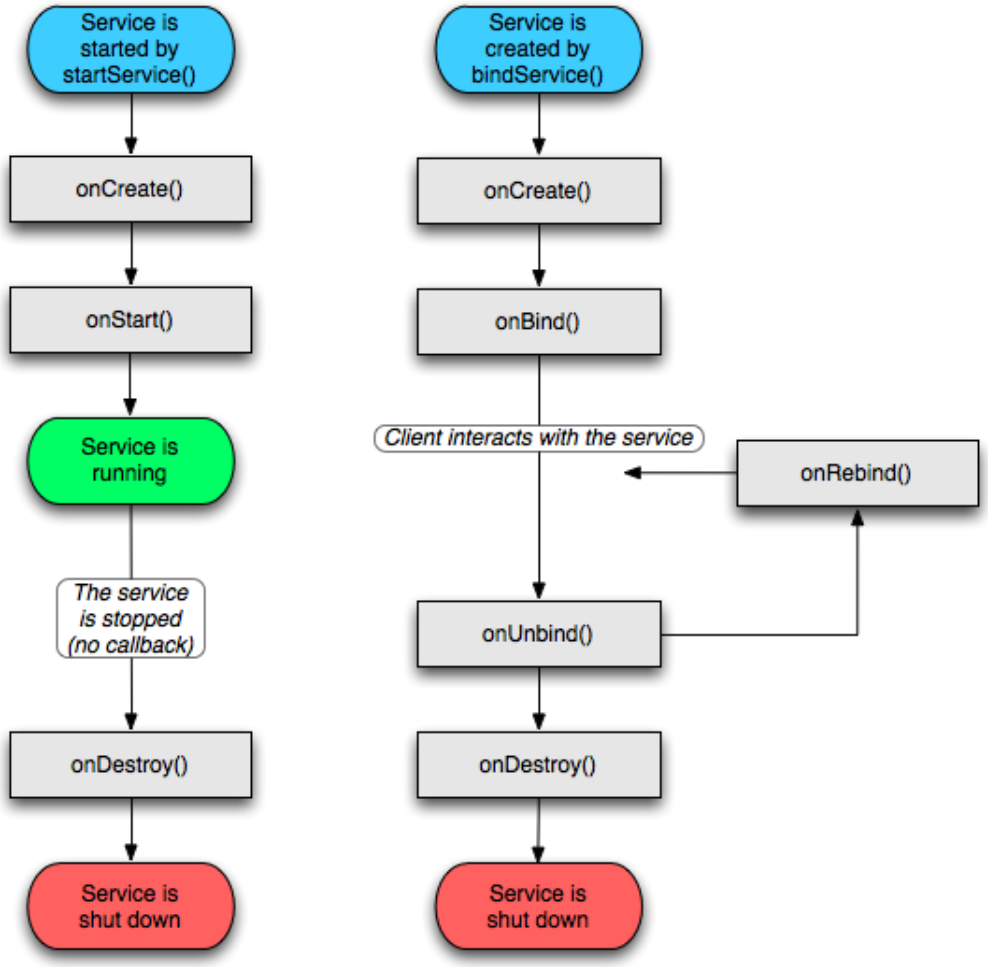
Pokud je Aktivita ve stavu *paused* nebo *stopped*, systém ji může vyzvat k ukončení voláním metody `finish()`, nebo ukončit přímo její proces. Pokud je Aktivita opět aktivována, musí být znovu spuštěna a musí obnovit svůj původní stav [1].

Služba

Služba může být využívána dvěma různými způsoby. Buď může být spuštěna zavoláním `Context.startService()` a běžet dokud není zastavena pomocí `Context.stopService()` nebo se nezastaví sama voláním `Service.stopSelf()` či `Service.stopSelfResult()`. Nebo



Obrázek 2.2: Životní cyklus Aktivity. Převzato z [1].



Obrázek 2.3: Životní cyklus Služby. Převzato z [1].

může být ovládána pomocí svého rozhraní a klienti se k ním připojují voláním metody `Context.bindService()`. `bindService()` spustí službu, pokud ještě neběží a ke službě se může současně připojit více klientů. Klienti se můžou samozřejmě ke službě připojovat i pokud byla spuštěna prvním způsobem [1].

Broadcast receiver

Pokud je poslána broadcastová zpráva a daný receiver je příjemcem, zavolá systém jeho metodu `onReceive()`. Tím se receiver aktivuje a je aktivní pouze po dobu zpracovávání této metody. Proces běžícího receiveru nemůže být ukončen. To může být problém, pokud je toto zpracování časově náročné. V takovém případě je možné pro zpracování spustit službu [1].

2.2.5 Ukládání dat

Android nabízí několik možností pro ukládání perzistentních dat. Výběr způsobu uložení závisí na povaze dat, na tom, zda mají být data přístupná jiným aplikacím a na velikosti místa, které potřebují.

- **Shared Preferences** – hodí se k ukládání malého množství dat. Je možné ukládat pouze primitivní datové typy (čísla, řetězce). Data jsou ukládána ve formě klíč–hodnota.
- **Interní úložiště** – data jsou uložena ve vnitřní paměti zařízení a standardně jsou nepřístupná ostatním aplikacím. Po odinstalování aplikace jsou tato data smazána.
- **Externí úložiště** – data uložena v externí paměti jsou přístupná uživateli a jiným aplikacím. Tato paměť může mít formu nevyjímatelné interní paměti, nebo nějakého externího média (v současnosti často SD karty). Tím vzniká nebezpečí, že data mohou být manipulací s tímto médiem smazána. Před čtením a zápisem do této paměti je třeba zjistit, jestli je právě přístupná. Pokud nejde přímo o data související s danou aplikací a uživatel chce předejít tomu, aby byla smazána při odinstalování aplikace, jsou k dispozici specifické adresáře pro ukládání těchto dat (`Music/`, `Pictures/`, ...).
- **SQLite databáze** – Android nabízí plnou podporu SQLite databází. Vytvořené databáze jsou přístupné všem třídám aplikace přes jejich jména, ale nejsou přístupná mimo aplikaci.
- **Síť** – využití vlastního serveru pro uchování dat.

Zdroje (Resources)

Externí data, která nejsou přímou součástí zdrojových kódů a je možné je separovat, je možné ukládat do zvláštního adresáře, který je pak součástí balíčku s aplikací. Jde především o soubory `xml` s popisem layoutu (vzhledu) aplikace, soubory s řetězci pro různé jazykové mutace aplikace, nebo obrázky. Přistupovat k těmto datům lze pouze pro čtení.

Některé zdroje jsou dány jako výchozí a jsou automaticky používány aplikací, pokud jsou přítomny. Uživatel například může specifikovat v adresáři `res/layout/` vzhled uživatelského prostředí. Pokud navíc vytvoří další vzhled a uloží jej do adresáře `res/layout-land/`, bude tento automaticky používán v *landscape* módu. Je ale také možné vytvářet vlastní zdroje a k nim z aplikace přistupovat [1].

Kapitola 3

Biometrie obličeje

Tato kapitola popisuje některé vlastnosti obličeje, které nás zajímají z hlediska biometrie. Dále jsou zde krátce probrány základní vlastnosti biometrických systémů.

3.1 Obličej z hlediska biometrie

Obličej je v biometrii často využíván, protože má poměrně dobré biometrické vlastnosti. Má vysokou univerzalitu a jde jednoduše nasnímat. Na druhou stranu vykazuje vysokou vnitrotřídní variabilitu (např. vliv emocí na vzhled obličeje, stárnutí osob) a v některých případech nízkou mezitřídní variabilitu – jednovaječná dvojčata.

Rozpoznávání obličeje je problém oboru biometrie, který se v tomto ohledu zabývá dvěma přístupy.

- Verifikace obličeje – tj. porovnání $1 : 1$. Dotyčná osoba podává tvrzení o své identitě a úkolem systém je určit, zda tato identita souhlasí, či ne.
- Identifikace obličeje – je porovnání $1 : N$, tedy proces složitější. Cílem je, aby systém určil identitu osoby. Přitom může nastat situace, že osoba v systému není registrována.

Úspěšnost algoritmů pro identifikaci ovlivňuje spousta faktorů a jedním z nich je kvalita pořízeného snímku. Vnitrotřídní variabilitu ovlivňují mimojiné následující vlastnosti:

- natočení hlavy,
- osvětlení,
- výraz obličeje,
- částečné zakrytí obličeje (brýlemi, módními doplňky, ...),
- vousy.

Vliv některých z nich lze omezit vhodným předzpracováním pořízených snímků.

3.2 Vlastnosti biometrických systémů

Výsledkem identifikace biometrického systému nemusí být přímo jednoznačná odpověď na identitu osobu, ale používá se tzv. míra shody (*matching score*). Ta udává podobnost mezi

nasnímanými vlastnostmi a hodnotami uloženými v databázi. Míra shody se většinou uvádí jako hodnota z intervalu $\langle 0, 1\rangle$, resp. procentuelně $\langle 0\%, 100\%\rangle$. Při verifikaci je potom přímá odpověď závislá na nastaveném prahu z tohoto intervalu. Pokud je zvolen příliš vysoký, hrozí odmítnutí právoplatného uživatele, při nízkém prahu může dojít povolení přístupu neoprávněné osobě [10].

V souvislosti s těmito chybami se používají některé ustálené pojmy: chybná shoda (*False Match*) – vzory dvou různých osob jsou označeny jako stejné, chybná neshoda (*False Non-Match*) – dva vzory jedné osoby jsou označeny jako různé. V praxi tedy můžou nastat čtyři situace [10]:

- správné přijetí (*True Acceptance*),
- správné odmítnutí (*True Rejection*),
- chybné přijetí (*False Acceptance*),
- chybné přijetí (*False Rejection*).

Jelikož je toto jedna z mála vlastností biometrických systémů, které lze měřit, používají se míry výskytu těchto situací k hodnocení jejich kvality [10].

Míra chybného přijetí FAR (*False Acceptance Rate*)

Pravděpodobnost, že systém chybně označí vzory různých osob jako shodné a přijme tak neoprávněnou osobu.

$$FAR = \frac{počet\ shodných\ porovnání\ rozdílných\ vzorů}{celkový\ počet\ porovnání\ rozdílných\ vzorů} \tag{3.1}$$

Míra chybného odmítnutí FRR (*False Rejection Rate*)

Pravděpodobnost, že systém chybně označí dva vzory jedné osoby jako různé a odmítne tak oprávněnou osobu.

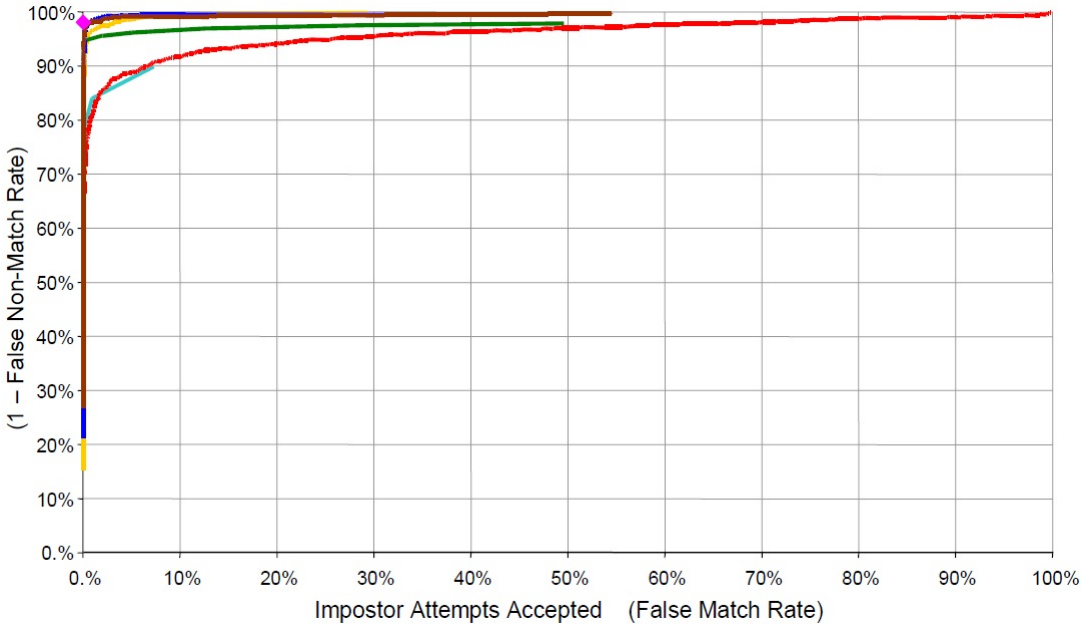
$$FRR = \frac{počet\ porovnání\ vzorů\ osoby\ A\ vedoucí\ k\ neshodě}{celkový\ počet\ porovnání\ vzorů\ osoby\ A} \tag{3.2}$$

Míra chybné shody (FMR) a Míra chybné neshody (FNMR)

FMR (*False Match Rate*) a FNMR (*False Non-Match Rate*) udávají podobně jako FAR a FRR počet chybně přijatých, resp. nepřijatých osob. Na rozdíl od nich ale neberou v úvahu chyby vzniklé ještě před porovnáním, tj. chyby vzniklé při snímáním.

ROC křivka

Výše uvedené míry nejsou příliš vhodné pro porovnávání systémů mezi sebou, jelikož se mění podle zvoleného prahu pro přijetí / odmítnutí. Byla proto zavedena tzv. ROC (*Receiver Operating Curve*) křivka, která je v současnosti standardem pro popis vlastností biometrického systému. Křivka je grafem závislosti FNMR (FRR) na FMR (FAR) [10].



Obrázek 3.1: Příklad ROC křivky. Převzato z [15].

3.3 Metriky

Pro výpočet vzdálenosti dvou vektorů se v biometrii používá několik metrik.

Vzdálenost *City block*

Známá i jako Manhattanská vzdálenost. Dráha z jednoho bodu do druhého v takovém prostoru vede pouze ve směrech rovnoběžných se souřadnicovými osami. Vzdálenost dvou bodů **r** a **s** v n-rozměrném prostoru je dána součtem absolutních hodnot rozdílu jednotlivých souřadnic těchto bodů, tedy [23]:

$$d_1(\mathbf{r}, \mathbf{s}) = \sum_{i=1}^{n} |r_i - s_i|.$$

Euklidovská vzdálenost

Klasická metrika pro výpočet vzdálenosti mezi dvěma body. Vzdálenost mezi dvěma body je dána odmocninou ze součtu druhých mocnin rozdílu jednotlivých souřadnic:

$$d_2(\mathbf{r}, \mathbf{s}) = \sqrt{\sum_{i=1}^{n} (r_i - s_i)^2}.$$

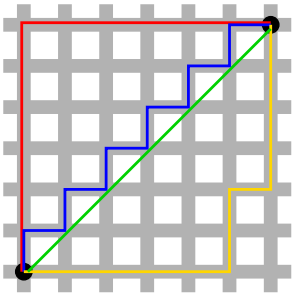
Metrika Minkowski

Jde o zobecnění předchozích dvou metrik, Euklidovské a Manhattan:

$$d_p(\mathbf{r}, \mathbf{s}) = \left(\sum_{i=1}^{n} |r_i - s_i|^p \right)^{\frac{1}{p}}.$$

Pro $p = 1$ jde tedy o metriku Manhattan, pro $p = 2$ o metriku Euklidovskou. V krajním případě, kdy se p blíží nekonečnu, jde o *Chebyshevovu vzdálenost*, která je rovna největší absolutní hodnotě rozdílů jednotlivých souřadnic. Bývá označována i jako *šachová vzdálenost* – při šachové hře jde o minimální počet tahů krále pro pohyb z jednoho pole na jiné [22]:

$$d_{Chebyshev}(\mathbf{r}, \mathbf{s}) = \lim_{p \to \infty} \left(\sum_{i=1}^{n} |r_i - s_i|^p \right)^{\frac{1}{p}} = \mathbf{max}_{i=1}^{n} |r_i - s_i|.$$



Obrázek 3.2: Porovnání Euklidovské a Manhattanské vzdálenosti. V případě Manhattanské může mezi dvěma body existovat několik nejkratších cest. U euklidovské je nejkratší cesta unikátní. Převzato z [23].

Mahalanobisova vzdálenost

Hodnota Mahalanobisovy vzdálenosti je závislá na hustotě rozložení dat a na bodu, ze kterého je pozorována. Oproti Euklidovské vzdálenosti je invariantní vůči změně měřítka [21].

$$d_m(\mathbf{r}, \mathbf{s}) = \sqrt{(\mathbf{r} - \mathbf{s})^T S^{-1} (\mathbf{r} - \mathbf{s})},$$

kde S je kovarianční matice.

Pokud je kovariantní matice jednotková, jde o klasickou Euklidovskou vzdálenost. Pokud je diagonální, jde o tzv. *normalizovanou Euklidovskou vzdálenost.*

Kapitola 4

Identifikace obličeje

Tato kapitola popisuje přístup ke zpracování obrazu z hlediska problematiky identifikace osob podle snímku obličeje. Jsou uvedeny některé postupy pro úpravu snímků, které mohou mít vliv na úspěšnost algoritmů pro identifikace. Následuje popis některých metod použitelných pro identifikaci.

4.1 Předzpracování obrazu

Kvalita vstupního snímku může být ovlivněna různými faktory – nerovnoměrným osvětlením, malým kontrastem apod. Tyto nedostatky lze odstranit vhodně zvolenými filtry. Nejčastěji se provádí právě redukce osvětlení a ekvalizace histogramu. Tím se zvýší úspěšnost následných metod pro identifikaci.

4.1.1 Knihovna OpenCV

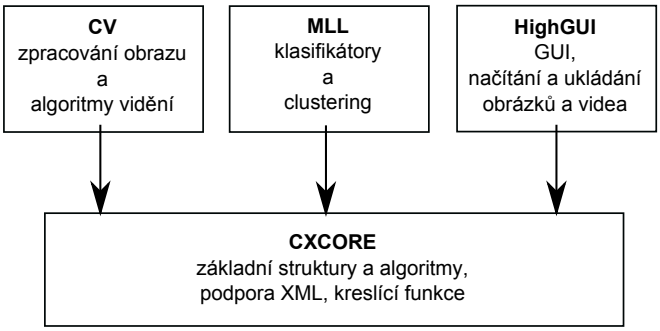
OpenCV je opensource knihovna počítačového vidění. Je napsaná v jazycích C a C++ a použitelná na operačních systémech Linux, Windows a Mac OS X. Knihovna je navržená pro realtime aplikace a je u ní kladen velký důraz na výpočetní efektivitu. Obsahuje více jak 500 funkcí od oblasti počítačového vidění, nejrůznějších obrazových filtrů, po algoritmy strojového učení [5].

Knihovna se skládá z několika hlavních částí:

- *CXCORE* – datové struktury, maticová algebra, transformace dat, správa paměti a chyb, ...
- *CV* – zpracování obrazu, analýza struktury obrazu, tracking, rozpoznávání vzorů, kalibrace kamery.
- *Machine Learning (ML)* – algoritmy spojené se strojovým učením, clustering, klasifikace, analýza dat.
- *HighGUI* – uživatelské rozhraní, ukládání / načítání obrázků a videa.

4.2 Rozdělení metod

Možností jak přistupovat k identifikaci existuje spousta. Liší se ve způsobu nahlížení na vstupní data. Některé berou v potaz obraz jako celek, jiné v něm vyhledávají určité části a



Obrázek 4.1: Struktura knihovny OpenCV [5].

hledají mezi nimi souvislosti. Autor [13] používá jednoduché rozdělení na metody *strukturální*, *holistické* a jejich kombinace.

Holistické metody identifikace

Holistické metody přistupují k obrazu jako k celku. Vstup je převeden do jiného prostoru (s menším počtem dimenzí, které nesou nejvíce informací). Používají se kombinace metod zpětného učení neuronové sítě (backpropagation), analýzy hlavních komponent (PCA) a singulární rozklad matice [13].

Strukturální metody identifikace

Strukturální přístup spočívá v lokalizaci některých významných částí obličeje a zjištění vztahu mezi nimi (vzájemné polohy). Výsledky měření je nutné znormalizovat, aby byly potlačeny nežádoucí vnější vlivy jako šum, různá poloha objektů vůči celku a jejich velikost. K určení podobnosti s identitami známých osob v databázi se pak používají různé klasifikátory. Z dominantních částí, které lze poměrně snadno detekovat, je možné zmínit hranici obličeje, oblast očí a úst, spodní část nosu, nebo místo, kde nos přechází v čelo.

V [13] je zmíněno, že pro detekci hranic obličeje je možné použít *Aktivní kontury* (známé jako *Snakes*), což je algoritmus pro segmentaci obrazu. Ve stručnosti jde o parametrickou křivku, která je deformována podle sil (resp. energií), které na ni působí. Vliv na výsledný tvar křivky mají tzv. vnější a vnitřní energie (vzájemné působení bodů křivky na sebe) a energie obrazu (vliv intenzity pixelů, resp. gradientu). Cílem iteračního postupu adaptace kontury na obraz je minimalizace součtu těchto energií.

Další možností je *Houghova transformace*. Při jejím použití se předpokládá, že tvar obličeje lze aproximovat elipsou. Může ovšem selhat, pokud se tvar od elipsy liší.

Jiným přístupem, je detekce podle barvy. Takto lze lokalizovat např. i oblast úst. Znalostí podprostoru barvy kůže, resp. rtů, je možné tyto oblasti zvýraznit transformací do jiného barevného prostoru. Následně můžeme použít jednoduchou segmentaci prahováním. Nevýhodou je nutnost znalosti informací o barvě, pro šedotónové snímky je tento postup nepoužitelný.

4.3 Holistické metody identifikace

V této části jsou popsány vybrané metody, které přistupují ke vstupnímu obrazu jako celku. Důležitou metodou je analýza hlavních komponent, která se často využívá v kombinaci s jinými kvůli redukci prostoru.

4.3.1 Analýza hlavních komponent (PCA)

Analýza hlavních komponent (Principal Component Analysis) je statistická metoda sloužící k redukci prostoru příznaků s co nejmenší ztrátou informace. Redukce tohoto prostoru umožňuje jednodušší reprezentaci dat, menší paměťové nároky a rychlejší klasifikaci [19].

Možnost využití analýzy hlavních komponent pro rozpoznávání obličeje popisuje Kyungnam Kim ve své práci [14], ze které čerpá následující text.

Myšlenka použití PCA v rozpoznávání osob podle obličeje je vyjádření velkého jednorozměrného vektoru pixelů, vzniklého z dvojrozměrného snímku obličeje, kompaktnějšími hlavními komponentami prostoru příznaků. Tomuto prostoru se říká *eigenspace* (vlastní prostor) a je možné ho vytvořit vypočítáním vlastních vektorů (*eigenvector*) kovarianční matice množiny 2D snímků obličeje.

Mějme množinu snímků obličejů, které jsou převedeny na jednorozměrný vektor tak, že řádky (případně sloupce) jsou zapsány za sebou. Tím vzniknou dlouhé jednorozměrné vektory. Dostáváme tedy M vektorů velikosti N (kde N = *(počet sloupců snímku)* $\times$ *(počet řádků snímku)*).

$$x_i = [p_1 \dots p_N]^T, i = 1, \dots, M \tag{4.1}$$

Tyto vektory upravíme tak, že od každého odečteme vektor průměrného obrazu m.

$$m = \frac{1}{M} \sum_{i=1}^{M} x_i \tag{4.2}$$

Výsledkem jsou vektory w_i.

$$w_i = x_i - m \tag{4.3}$$

Cílem je najít množinu M ortonormálních vektorů e_i, pro něž je

$$\lambda_i = \frac{1}{M} \sum_{i=1}^{M} (e_i^T w_n)^2 \tag{4.4}$$

největší. Přitom e_i a λ_i jsou vlastní vektory a vlastní čísla kovarianční matice

$$C = WW^T, \tag{4.5}$$

kde W je matice tvořená sloupcovými vektory w_i. Dimenze takové matice může být ovšem v tomto případě obrovská ($N \times N$), např. pro snímky velikosti 64×64 má kovarianční matice rozměry 4096×4096. Z lineární algebry vyplývá, že vektory e_i a čísla λ_i mohou

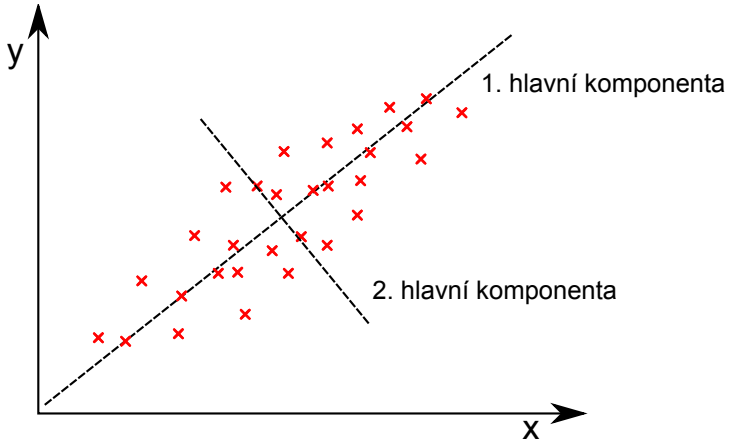
být získány jako vlastní vektory a čísla matice $W^T W$ velikosti $M \times M$. Nechť jsou d_i a μ_i vlastní vektory a vlastní čísla matice $W^T W$, tedy

$$W^T W d_i = \mu_i d_i. \tag{4.6}$$

Vynásobení obou stran W dostaneme

$$W W^T (W d_i) = \mu_i (W d_i). \tag{4.7}$$

To znamená, že prvních $M - 1$ vlastních vektorů e_i a vlastních čísel λ_i matice $W W^T$ je dáno $W d_i$ a μ_i. Jelikož snímků je konečný počet M hodnost kovarianční matice není větší než $M - 1$ (-1 protože od vektorů byl odečten průměrný snímek).



Obrázek 4.2: Hlavní komponenty množiny bodů v 2D prostoru.

Vlastní vektory, jejichž odpovídající vlastní hodnoty jsou nenulové, tvoří ortonormální bázi podprostoru, ve kterém mohou být data snímků reprezentována bez větších chyb. Vlastní vektory jsou seřazené podle jim odpovídajících vlastních hodnot od největší po nejmenší. Čím větší je vlastní hodnota, tím větší je změna dat ve směru daného vektoru. Hodnota vlastních čísel klesá exponenciálně. To znamená, že prvních 5 až 10% hodnot vektorů nese 90% informace o změně v obraze.

Snímek obličeje může být převeden do prostoru dimenze $M' (\ll M)$ jako:

$$\Omega = [v_1 v_2 \dots v_{M'}]^T, \tag{4.8}$$

kde $v_i = e_i^T w_i$ je i–tá souřadnice snímku v novém prostoru. Vektory e_i bývají v tomto kontextu nazývány *eigenfaces*.

Rozpoznávání obličeje analýzou hlavních komponent pak probíhá následovně. Z trénovacích dat (snímků obličejů) je vytvořena báze prostoru, do kterého jsou snímky promítány. Výsledkem projekce do tohoto prostoru je vektor hodnot, který udává váhy jednotlivých eigenfaces. Pokud si uložíme tyto vektory do paměti, můžeme je potom použít pro identifikaci osoby ze snímku, který systém zatím neviděl. Pokud dostane systém na vstup neznámý obličej a převede ho do prostoru hlavních komponent, může vektor vah porovnat s uloženými hodnotami. Tím může nastat několik situací:

Obrázek 4.3: Ukázka průměrného obličeje. Převzato z [24].

Obrázek 4.4: Příklad vzhledu eigenfaces. Převzato z [24].

- Vektor leží daleko od prostoru obličejů – zřejmě nejde o snímek s obličejem.
- Vektor leží v prostoru obličejů, ale není blízko žádné třídy – jde o neznámou osobu.
- Vektor leží v prostoru obličejů blízko již uloženého obličeje – jde o snímek téže osoby.

4.3.2 Analýza nezávislých komponent (ICA)

Analýza nezávislých komponent (Independent component analysis) se snaží separovat ze směsice signálů nezávislé komponenty. Je blízká problému slepé separace zdrojů (*BSS - blind source sparation*), jehož cílem je rozložení signálu na lineární kombinaci nezávislých signálů.

Mějme vektor $\mathbf{s}$, který je neznámým zdrojem, vektor $\mathbf{x}$, který je směsicí signálů, a neznámou směšovací matici A, popisující způsob smíchání signálů $\mathbf{s}$ na změřený signál $\mathbf{x}$. Můžeme pak popsat směšovací model jako:

$$\mathbf{x} = A\mathbf{s}. \tag{4.9}$$

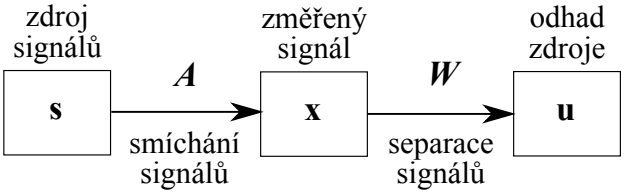
Matice A musí být invertibilní (regulární) a signály $\mathbf{s}$ jsou navzájem nezávislé. Algoritmus ICA se snaží najít pro vektor $\mathbf{x}$ směšovací matici A, nebo separační matici W a získat tak odhad zdroje nezávislých signálů u:

$$\mathbf{u} = W\mathbf{x} = WA\mathbf{s}. \tag{4.10}$$

Je snahou, aby získané signály u byly statisticky nezávislé, tzn.:

$$f_u(u) = \prod_i f_{u_i}(u_i), \tag{4.11}$$

kde f_u je funkce hustoty pravděpodobnosti (tj. aby data byla rozložena rovnoměrně). Bohužel taková matice W, která by splňovala podmínku nezávislosti, nemusí existovat. Existují ale iterační algoritmy pro aproximaci matice W, aby byla nezávislost co nejvyšší.

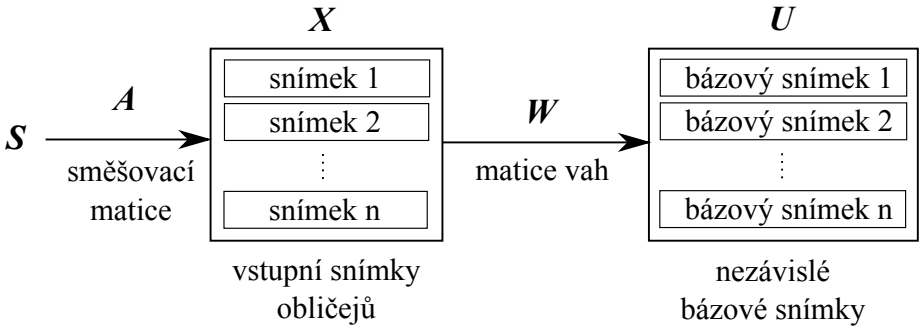


Obrázek 4.5: Model slepé separace zdrojů. Upraveno z [11].

Při identifikaci se používá jeden ze dvou přístupů v závislosti na algoritmu zvoleném pro výpočet nezávislých komponent.

Statisticky nezávislé bázové snímky

V tomto případě jsou vstupní snímky obličejů X brány jako kombinace statisticky nezávislých bázových snímků S po sloučení pomocí směšovací matice A. Pomocí ICA algoritmu je získána váhová matice W, která je použita k získání sady nezávislých bázových snímků U. Projekce vstupního snímku pomocí váhové matice dostáváme bázový snímek. Snímek obličeje je komprimován do sady koeficientů, použitých k vytvoření daného snímku lineární kombinací bází.



Obrázek 4.6: Nalezení statisticky nezávislých bázových snímků. Upraveno z [11].

Existují varianty tohoto postupu, kdy je na snímky nejdříve aplikována analýza hlavních komponent (4.3.1), kvůli zredukování dimenzí prostoru snímků a tím pádem i snížení počtu nezávislých komponent produkovaných ICA.

Mějme matici R velikost $p \times m$, obsahující prvních m vlastních vektorů sady n obličejových snímků a kde p je počet pixelů trénovacích dat. Na matici R^T je použita ICA a m nezávislých bází uložených jako řádky matice U je spočítáno jako $U = WR^T$. Potom $m \times n$ koeficientů matice B pro lineární kombinaci nezávislých bázových snímků v U je spočítáno následovně [11]:

$$C = XR, \qquad X = CR^T, \tag{4.12}$$

kde C je matice velikosti $n \times m$ obsahující PCA koeficienty. Z $U = WR^T$ a faktu, že W je invertibilní, vyplývá:

$$R^T = W^{-1}U. \tag{4.13}$$

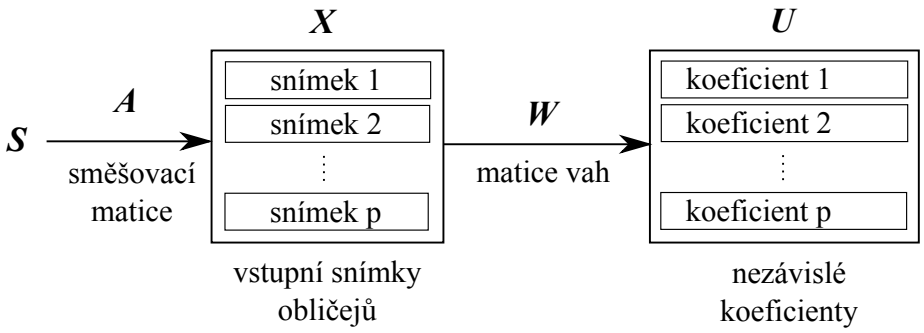
A tedy

$$X = \left(CW^{-1}\right)U = BU. \tag{4.14}$$

Každý řádek matice B obsahuje koeficienty pro lineární kombinaci bázových snímků, tvořící snímek obličeje v odpovídajícím řádku matice X.

Statisticky nezávislé koeficienty

Zatímco bázové snímky v předchozím případě jsou statisticky nezávislé, koeficienty, reprezentující vstupní obrázek v podprostoru definovaném bázovými snímky závislé jsou. Cílem tohoto přístupu je najít statisticky nezávislé koeficienty pro vstupní data. Vstup je narozdíl od předchozího případu transponovaný. Separace zdrojů se provádí na pixelech a každý řádek váhové matice W je snímkem. Matice A, inverzní k matici W, obsahuje ve sloupcích bázové snímky. Statisticky nezávislé koeficienty zdrojů v S, které vytváří vstupní snímky, jsou získány jako sloupce matice U. Podobně jako v předchozím případě se používá pro předzpracování PCA, pro snížení dimenze dat [11].



Obrázek 4.7: Nalezení statisticky nezávislých koeficientů. Upraveno z [11].

4.3.3 Lineární diskriminační analýza (LDA)

LDA vytváří na základě množiny trénovacích dat pravidla, která slouží ke klasifikaci dalších objektů do konečného počtu tříd. Cílem je vytvoření *rozptylové matice*, která popisuje rozdělení objektů do M tříd tak, aby rozdíly objektů uvnitř tříd byly co nejmenší a mezi třídami co největší. Výpočet vnitrotřídní (S_w) a mezitřídní (S_b) rozptylové matice je dán následovně [25]:

$$S_w = \sum_{i=1}^{M} Pr\left(C_i\right)\Sigma_i, \tag{4.15}$$

$$S_b = \sum_{i=1}^{M} Pr\left(C_i\right)\left(m_i - m_0\right)\left(m_i - m_0\right)^T, \tag{4.16}$$

kde $Pr(C_i)$ je pravděpodobnost příslušnosti ke třídě a v praxi bývá většinou nahrazena $\frac{1}{M}$. S_w je vnitrotřídní rozptylová matice, udávající střední odchylku Σ_i vektoru vzorků $\mathbf{x}$ různých tříd C_i okolo jejich středních hodnot m_i:

$$\Sigma_i = E\left[(x - m_i)(x - m_i)^T | C = C_i\right]. \tag{4.17}$$

Podobně S_b je mezitřídní rozptylová matice popisující rozptyl středních vektorů m_i okolo celkového středního vektoru m_0. K vyjádření diskriminační síly je využíváno několik technik, např. poměr determinantů mezi a vnitrotřídní rozptylové matice projektovaných vzorků:

$$\tau(T) = \frac{|T^T S_b T|}{|T^T S_w T|}. \tag{4.18}$$

Optimální projekce maximalizuje $\tau(T)$ podle W a to můžeme získat z následujícího:

$$S_b W = S_w W \Lambda_W. \tag{4.19}$$

K vyřešení tohoto problému je možné použít Choleskyho dekompozici, nebo nejprve vyřešit vlastní problém pro matici S_w. Protože S_w je reálná symetrická matice, existuje ortonormální Q a diagonální Λ_Q takové, že $S_w = Q\Lambda_Q Q^T$. Dostáváme tedy:

$$\left(\Lambda_Q^{-\frac{1}{2}} Q^T\right) S_b W = \left(\Lambda_Q^{\frac{1}{2}} Q^T\right) W \Lambda_W, \tag{4.20}$$

což lze zjednodušit na:

$$\left(R S_b R^T\right) W_R = W_R \Lambda_W, \tag{4.21}$$

kde $R = \Lambda_Q^{-\frac{1}{2}} Q^T$ a $W_R = \Lambda_Q^{\frac{1}{2}} Q^T W$. Vyřešení (4.21) získáme Λ_W a W_R a můžeme spočítat W jako $Q\Lambda_Q^{-\frac{1}{2}} W_R$.

Při identifikaci obličeje bývá velikost matice obrovská a matice S_W má díky zaokrouhlovací chybě blízko k singulární. K vyřešení tohoto problému upravíme matici S_W na $S_W - \Delta I$, kde Δ je relativně malé kladné číslo. Tento postup se liší od tradiční redukce dimenze pomocí PCA. Oba tyto přístupy se ale potýkají s otázkou, jak postupovat, pokud je počet trénovacích vzorků menší než velikost vektoru snímku. Obvyklá redukce dimenze neumí vyřešit problém, kdy je pro každou třídu k dispozici pouze jeden trénovací vzorek. V tom případě je S_W rozšířena na matici identity, takže problém je převeden na klasický výpočet vlastních hodnot [25].

4.4 Statistické modely

Tyto modely mají společné některé znaky – trénovací sada je vytvořena ze vzorků, získaných statistických sběrem dat. Problémem může být ale automatizované získání relevantních dat, proto bývá pro tento účel potřeba manuální anotace. Získané vzorky jsou zarovnány, normovány a je z nich vytvořen model. Změnou parametrů modelu vznikají nové objekty stejné třídy, jako objekty trénovací sady. Iteračním postupem je potom model adaptován na daný vstup.

4.4.1 Point distribution model (PDM)

PDM je technika pro popis tvaru, která může být použita ke generování nových instancí tvarů objektů stejné třídy. Popisuje vlastnosti obecného tvaru v případech, kdy není možné použít rigidní model. PDM vychází z trénovací sady M vzorků, z kterých je odvozen statistický popis tvaru a jeho změny. Je vybráno N bodů na hranicích objektů, které jsou označeny na každém vzorku. Tohoto přístupu se používá u dalších metod popsaných níže [18].

4.4.2 Active Shape Model (ASM)

Tato metoda vychází ze statistického modelu tvaru [9], který je používán k reprezentaci objektů v obraze. Tvar objektu je reprezentován množinou n bodů a je invariantní vůči některým transformacím – rotaci, posunutí a změně měřítka. Body nemusí být nutně z dvojdimenzionálního prostoru (většinou to tak je) ale obecně z prostoru dimenze d. Trénovací sada většinou vychází z ruční anotace sady trénovacích snímků (i když jsou ve vývoji automatické systémy). Analýzou tvarů z trénovací množiny je vytvořen statistický model.

Vhodné body pro popis tvaru jsou místa, která se nachází napříč všemi snímky. Vhodná místa jsou hranice objektů, T–spoje mezi hranicemi, rohy a podobné, dobře rozlišitelné body. Další body je možné umístit v rovnoměrné vzdálenosti mezi těmito místy. Nejjednodušší cestou k vytvoření testovací sady je člověk, který tyto body manuálně označí na všech testovacích snímcích. Pokud máme tvar popsaný n body (x_i, y_i), můžeme ho zapsat jako vektor x délky $2n$ [9]:

$$x = (x_1, \ldots, x_n, y_1, \ldots, y_n)^T. \tag{4.22}$$

Máme–li s trénovacích dat, pak takto můžeme vytvořit s vektorů x_j. Před tím, než budeme moct tato data statisticky analyzovat, je třeba aby všechny tvary byly ve stejném souřadnicovém rámci. Jednou z možností jak toho dosáhnout je použití *Procrustovy analýzy*. Ta zarovná každý tvar tak, že součet všech vzdáleností tvarů od průměru ($D = \sum |x_i - \overline{x}|^2$) je minimalizován. Iterativní algoritmus pro toto zarovnání vypadá následovně [9]:

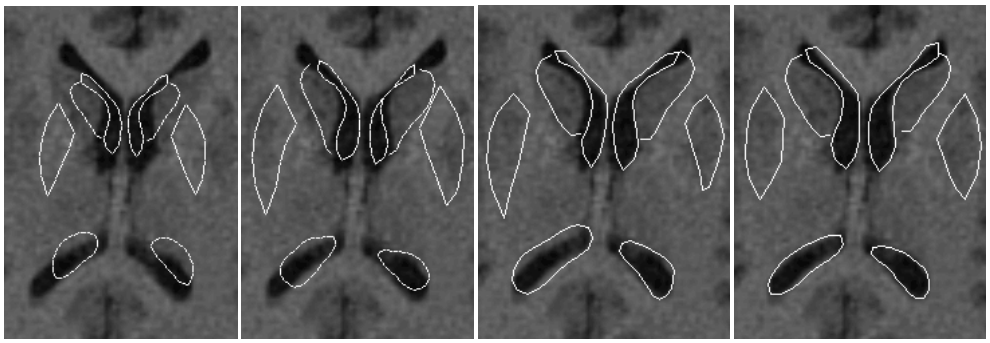
Algoritmus 4.4.1. Zarovnání trénovacích dat modelu ASM:

1. Posuň každý tvar tak, aby měl těžiště v počátku.
2. Vyber jeden tvar jako výchozí průměrný tvar a změň měřítko tak, aby $|\overline{x}| = 1$.
3. Definuj výchozí referenční rámec – nastav první odhad jako $\overline{x}_0$.
4. Zarovnej všechny tvary s aktuálním odhadem průměrného tvaru.
5. Přepočítej střední tvar ze zarovnaných tvarů.
6. Uplatni omezení na aktuální odhad jeho zarovnáním s $\overline{x}_0$ a změnou měřítka tak, aby $|\overline{x}| = 1$.
7. Pokud se odhad významně změní, pokračuj bodem 4, jinak skonči.

Pokud nyní máme s množin bodů x_i, které jsou zarovnané do společné souřadnicové soustavy, můžeme generovat nové tvary, podobné těm z trénovací sady. Budeme hledat parametrizovaný model M, $x = M(b)$, tedy model, který na základě vektoru parametrů b vygeneruje vektor x. Kvůli zjednodušení výpočtu je možné změnit dimenzi dat z nd na menší např. algoritmem PCA.

Algoritmus 4.4.2. Iterativní postup pro nalezení modelu odpovídajícímu tvaru objektu na snímku [9]:

1. Prozkoumej oblast obrázku v okolí bodu $\mathbf{X}_i$ a najdi nejlepší blízkou shodu pro bod $\mathbf{X}'_i$.
2. Aktualizuj parametry $(X_t, Y_t, s, \theta, \mathbf{b})$.
3. Opakuj postup, dokud konverguje.



Obrázek 4.8: Příklad konvergence modelu k objektu na obrázku (zleva: výchozí stav, po 1 iteraci, po 6 iteracích, po 12 iteracích). Převzato z [8].

V praxi např. předpokládáme, že hranice modelu odpovídají hranám v obrázku. Můžeme tedy jednoduše lokalizovat nejsilnější hranu podél profilu k získání nové pozice bodu [8].

4.4.3 Active appearance model (AAM)

Stejně jako v předchozím případě jde o statistický model vycházející z PDM. Narozdíl od Active Shape modelu, který manipuluje pouze s tvarem modelu, využívá i další informaci získatelnou ze zdrojového obrázku, texturu. Model je vytvořen tak, že testovací vzorky jsou warpingem transformovány na průměrný tvar. To samozřejmě vyžaduje, aby vyznačené body mezi jednotlivými vzorky co nejlépe korespondovaly. Pomocí analýzy hlavních komponent – zvlášť pro model tvaru a zvlášť pro model popisující intenzitu pixelů textury – je možné vytvářet nové objekty. Segmentace AAM je minimalizace rozdílu mezi vygenerovaným modelem intenzit a cílovým obrázkem. Tento rozdíl je možné jednoduše počítat například jako součet druhých mocnin rozdílu v intenzitě jednotlivých pixelů [18].

Algoritmus 4.4.3. Vytvoření AAM [18]:

1. Vypočti ASM a aproximuj každý vzor tvaru jako lineární kombinaci vlastních vektorů, $\mathbf{b}_s = P_s^T(\mathbf{x} - \overline{\mathbf{x}})$ reprezentuje parametry vzoru tvaru.
2. Pomocí warpingu transformuj každý vzorový snímek do průměrného tvaru (za použití lineární nebo nelineární interpolace).
3. Normalizuj každý vzorový snímek podle průměrné intenzity a rozptylu $\overline{\mathbf{g}}$.
4. Použij PCA na normalizované snímky.

5. Vyjádři každý vzorový snímek jako lineární kombinaci vlastních vektorů, $\mathbf{b}_g = P_g^T(\mathbf{g} - \overline{\mathbf{g}})$ reprezentuje parametry vzorku textury.

6. Proveď konkatenaci vektorů, reprezentujících parametry tvaru a textury, pomocí následujícího vztahu:
$$\mathbf{b} = \begin{bmatrix} W\mathbf{b}_s \\ \mathbf{b}_g \end{bmatrix} = \begin{bmatrix} WP_s^T(\mathbf{x} - \overline{\mathbf{x}}) \\ P_g^T(\mathbf{g} - \overline{\mathbf{g}})\mathbf{b}_g \end{bmatrix},$$
kde W je diagonální váhová matice, která upravuje váhu parametrů tvaru a textury.

7. Aplikuj PCA na vzorovou sadu všech vektorů b. Výsledkem je model
$$\mathbf{b} = Q\mathbf{c},$$
kde Q je matice složená z vlastních vektorů a $\mathbf{c}$ jsou koeficienty, které charakterizují, jak se výsledný model liší od průměrného modelu. Pokud jsou prvky vektoru $\mathbf{c}$ nulové, je výsledkem model průměrného tvaru a textury.

Obrázek 4.9: Příklad modelu vzhledu (Dr. Tim Cootes). Uprostřed je průměrný vzhled, vlevo a vpravo různé výsledky pro změněné parametry vektoru $\mathbf{c}$. Převzato z [6].

Pro nalezení nejlepší shody mezi modelem a obrázkem je třeba minimalizovat velikost rozdílového vektoru $|\Delta\mathbf{I}|^2$ změnami parametrů modelu. Rozdílový vektor $\delta\mathbf{I}$ může být definován následovně:

$$\delta\mathbf{I} = \mathbf{I_i} - \mathbf{I_m}, \tag{4.23}$$

kde $\mathbf{I_i}$ je vektor hodnot úrovní šedi v obrázku a $\mathbf{I_m}$ je vektor hodnot pro současné parametry modelu [7].

Máme–li aktuální odhad parametrů modelu c_0 a normalizovaný vstupní obraz g_s, můžeme přistoupit k iteračnímu cyklu pro úpravy modelu.

Algoritmus 4.4.4. Jeden krok při iteračním cyklu adaptace modelu:

1. Vyhodnoť chybový vektor $\delta g_0 = g_s - g_m$.

2. Vyhodnoť aktuální chybu $E_0 = |\delta g_0|^2$.

3. Vypočti nové hodnoty parametrů $\delta c = A\delta g_0$.
4. Nastav $k = 1$.
5. $c_1 = c_0 - k\delta c$.
6. Vytvoř nový obrázek podle nových parametrů a vypočti nový chybový vektor δg_1.
7. Pokud je $|\delta g_1|^2 < E_0$, zachovej nové parametry c_1.
8. Jinak opakuj postup pro $k = 1.5$, $k = 0.5$, $k = 0.25$, atd.

Tento postup je aplikován, dokud se chyba $|\delta g|^2$ mění a algoritmus konverguje.

Obrázek 4.10: Příklad adaptace modelu AAM. Zleva: výchozí stav, po 2, 8, 14, 20 iteracích a konečný stav. Převzato z [9].

Warping

Zásadní součástí adaptace modelu je algoritmus pro warping. Tvary obličejů jsou složeny z bodů, které jsou spojeny do trojúhelníků. Ke změně tvaru se tedy používá afinní warping. Každý bod $\mathbf{x}$ výchozího tvaru s_0 leží v trojúhelníku. Vrcholy tohoto trojúhelníku označme $(x_i^0, y_i^0)^T$, $(x_j^0, y_j^0)^T$ a $(x_k^0, y_k^0)^T$. Vrcholy odpovídajícího trojúhelníku v novém tvaru označme $(x_i, y_i)^T$, $(x_j, y_j)^T$ a $(x_k, y_k)^T$. Bod uvnitř $\mathbf{x} = (x, y)^T$ můžeme vyjádřit jako [16]:

$$\mathbf{x} = (x, y)^T = (x_i^0, y_i^0)^T + \alpha\left[(x_j^0, y_j^0)^T - (x_i^0, y_i^0)^T\right] + \beta\left[(x_k^0, y_k^0)^T - (x_i^0, y_i^0)^T\right], \quad (4.24)$$

kde

$$\alpha = \frac{(x - x_i^0)(y_k^0 - y_i^0) - (y - y_i^0)(x_k^0 - x_i^0)}{(x_j^0 - x_i^0)(y_k^0 - y_i^0) - (y_j^0 - y_i^0)(x_k^0 - x_i^0)}, \quad (4.25)$$

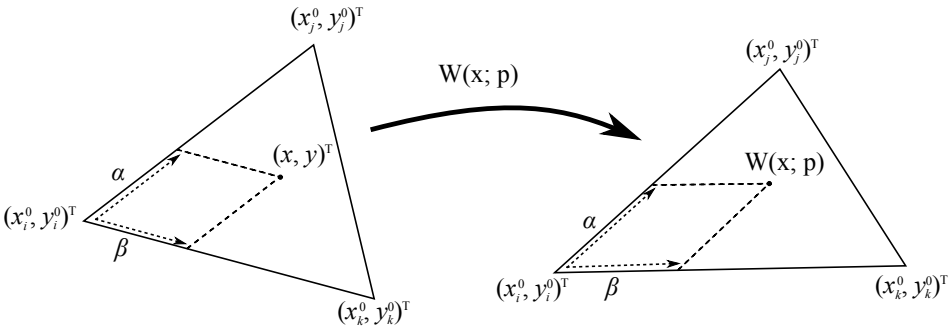
$$\beta = \frac{(y - y_i^0)(x_j^0 - x_i^0) - (x - x_i^0)(y_j^0 - y_i^0)}{(x_j^0 - x_i^0)(y_k^0 - y_i^0) - (y_j^0 - y_i^0)(x_k^0 - x_i^0)}. \quad (4.26)$$

Výsledek aplikace afinního warpingu je pak:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = (x_i, y_i)^T + \alpha\left[(x_j, y_j)^T - (x_i.y_i)^T\right] + \beta\left[(x_k, y_k)^T - (x_i, y_i)^T\right], \quad (4.27)$$

kde $(x_i, y_i)^T$, $(x_j, y_j)^T$ a $(x_k, y_k)^T$ jsou souřadnice vrcholů odpovídajícího nového trojúhelníku a $\mathbf{p}$ jsou parametry tvaru. Rovnice 4.25, 4.26 a 4.27 dávají dohromady jednoduchý afinní warping:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = (a_1 + a_2 x + a_3 y, a_4 + a_5 x + a_6 y)^T. \tag{4.28}$$



Obrázek 4.11: Warping trojúhelníku při adaptaci AAM. [16].

Parametry $(a_1, a_2, a_3, a_4, a_5, a_6)$ mohou být spočítány jednoduše z parametrů tvaru. Tento výpočet stačí provést pouze jednou pro každý trojúhelník, nemusí se počítat pro každý bod. Efektivní výpočet warpingu pak může vypadat následovně [16]:

Algoritmus 4.4.5. Warping tvaru AAM.

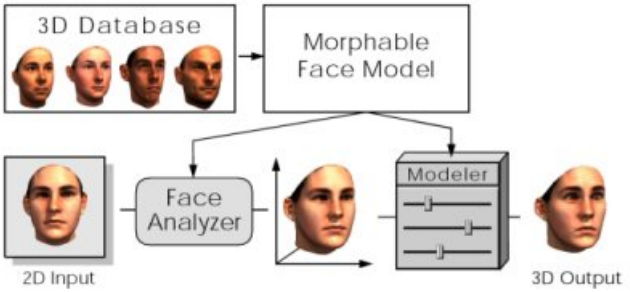
1. Pro parametry $\mathbf{p}$ spočítej vrcholy $(x_i, y_i)^T$ nového tvaru s.
2. Spočítej $(a_1, a_2, a_3, a_4, a_5, a_6)$ pro každý trojúhelník.
3. Pro každý pixel $\mathbf{x}$ tvaru s_0 najdi trojúhelník, ve kterém $\mathbf{x}$ leží a získej tak odpovídající hodnoty $(a_1, a_2, a_3, a_4, a_5, a_6)$.
4. Spočítej $\mathbf{W}(\mathbf{x}; \mathbf{p})$ pomocí rovnice 4.28.

4.4.4 3D modifikační model (3d morphable model)

Volker Blanz a Thomas Vetter představili v [3] metodu, která umožňuje uživateli modelovat velké množství různých obličejů jednoduchou změnou několika koeficientů. Navíc tento způsob umožňuje ve spojení s analýzou 2D snímku téměř plně automatizovanou rekonstrukci obličeje. Modifikační model vychází z databáze 3D scanů reálných obličejů, přičemž nové jsou vytvářeny jako jejich lineární kombinace. To dává k dispozici velké množství kombinací, které se přitom nevymykají přirozenému vzhledu.

Vytvoření databáze

K vytvoření databáze je využito několik desítek až stovek 3D scanů hlav dospělých osob obou pohlaví. Všechny osoby mají obličej prostý jakýchkoliv doplňků, vousů, či makeupu. Dalším krokem je vertikální (zadní část hlavy) a horizontální (ramena) ořezání a normalizace, která zajistí, že každý obličej bude mít stejnou orientaci a pozici v prostoru. Výsledné obličeje jsou reprezentovány polygonálním modelem s přibližně 70000 vrcholy a stejným počtem barevných hodnot. Společně s tvarem je tedy uložena i informace o barvě [3].



Obrázek 4.12: Schéma využití modifikačního modelu pro rekonstrukci obličeje z 2D snímku [3].

Vytvoření modelu

Modifikační model je založen na sadě trojrozměrných modelů. Aby bylo možné vytvářet jejich lineární kombinace, je třeba, aby spolu tyto modely svou strukturou plně korespondovaly. Geometrie tváře je reprezentována vektorem $S = (x_1, y_1, z_1, x_2, \ldots, x_n, y_n, z_n)^T \in \mathcal{R}^{3n}$, který popisuje souřadnice x, y a z každého z n vrcholů. Pro jednoduchost je počet barev stejný jako počet vrcholů (údaje o barvě jsou uchovávány pro každý vrchol). Textura obličeje je tedy reprezentována vektorem $T = (R_1, G_1, B_1, R_2, \ldots, R_n, G_n, B_n)^T \in \mathcal{R}^{3n}$, nesoucím informaci o RGB hodnotě každého vrcholu [3].

Nový model tvaru (S_{model}) a textury (T_{model}) obličeje dostaneme lineární kombinací vzorů z databáze:

$$S_{model} = \sum_{i=1}^{m} a_i S_i, \quad T_{model} = \sum_{i=1}^{m} b_i S_i, \tag{4.29}$$

kde

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} b_i = 1. \tag{4.30}$$

Modifikační model je tedy definován jako množina obličejů ($S_{model}(\mathbf{a})$, $T_{model}(\mathbf{b})$) parametrizovaných koeficienty $\mathbf{a} = (a_1, a_2, \ldots, a_m)^T$ a $\mathbf{b} = (b_1, b_2, \ldots, b_m)^T$. Nové obličeje jsou generovány změnou těchto parametrů, které ovlivňují tvar a texturu modelu [3].

Variabilitu modelu lze zvýšit rozdělením obličeje do několika částí (např. oči, nos, ústa a jejich okolí), které budou transformovány nezávisle. Tato segmentace je ekvivalentní rozdělení vektorového prostoru obličejů na nezávislé podprostory [3].

Algoritmus pro renderování scény

Výpočet syntetizované scény lze rozdělit na dvě části. První řeší pozici a natočení modelu (vrcholů modelu) a jeho perspektivní projekci. Souřadnice každého vrcholu k, $\mathbf{x}_k = (x_k, y_k, z_k)^T$ jsou transformovány do souřadnic relativních vůči kameře:

$$(w_{x,k}, w_{x,k}, w_{x,k})^T = \mathbf{R}_\gamma \mathbf{R}_\theta \mathbf{R}_\phi \mathbf{x}_k + \mathbf{t}_w, \tag{4.31}$$

kde úhly θ a ϕ definují rotaci kolem vertikální a horizontální osy a úhel γ rotaci kolem osy kamery. $\mathbf{t}_w$ je posunutí v prostoru [4].

Perspektivní projekce potom transformuje souřadnice vrcholů do 2D prostoru v závislosti na ohniskové vzdálenosti (f) [4].

$$p_{x,k} = P_x + f\frac{w_{x,k}}{w_{z,k}}, \qquad p_{y,k} = P_y + f\frac{w_{y,k}}{w_{z,k}} \tag{4.32}$$

Druhá část algoritmu řeší barvu a nasvětlení modelu. Stínování modelu závisí na směru normálových vektorů trojúhelníků, které tvoří model, a parametrech světel ve scéně. Výpočet osvětlení je řešen pomocí *Phongova osvětlovacího modelu*. Tento model poskytuje dostatečně přirozené výsledky a je relativně rychlý. Viditelnost vrcholů a stíny jsou řešeny pomocí z-bufferu.

Využití modelu k identifikaci

Blanz a Vetter popisují v [4] způsob využití modelu pro identifikaci osob. Pro každou osobu jsou z jednoho snímku získány vektory koeficientů $\mathbf{a} = (a_1, \ldots, a_{99})^T$ a $\mathbf{b} = (b_1, \ldots, b_{99})^T$ pro celý obličej a další vektory pro čtyři různé segmenty, na které je obličej rozdělen. Těchto pět dvojic vektorů je spojeno do jednoho vektoru $c \in \mathbb{R}^{990}$, přičemž koeficienty vektorů jsou poděleny standardními odchylkami $\sigma_{S,i}$ a $\sigma_{T,i}$.

Porovnání dvou vektorů $\mathbf{c}_1$ a $\mathbf{c}_2$ se provádí výpočtem *Mahalanobisovy vzdálenosti* (3.3). Alternativní možností je výpočet kosinu úhlu mezi těmito vektory $d = \frac{\langle \mathbf{c_1}, \mathbf{c_2} \rangle}{\|\mathbf{c_1}\| \cdot \|\mathbf{c_2}\|}$.

Výhodou použití tohoto modelu při identifikace je schopnost rozpoznat obličej i pokud je nasnímán pod jiným úhlem, než byl nasnímán trénovací snímek.

4.5 Skryté Markovovy modely (HMM)

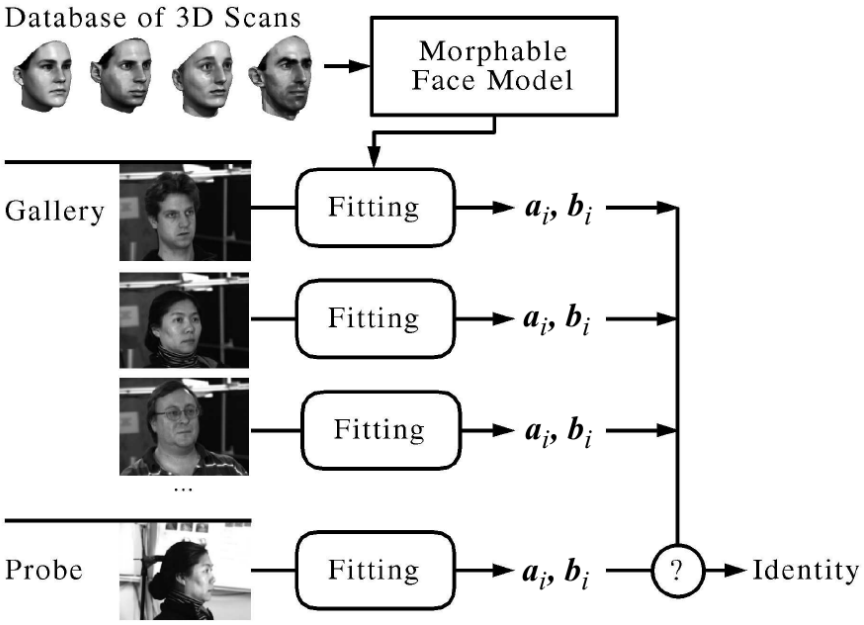
Skryté Markovovy modely (Hidden Markov models) je metoda většinou používaná při rozpoznávání řeči, A.M.Nefiana a M.H.Hayese [17] ovšem napadlo vyzkoušet ji i pro rozpoznávání obličeje. Jde o sadu statistických modelů, použitých k charakterizování statistických vlastností signálu. Modely jsou popsány pomocí tzv. Markovova řetězce s konečným počtem stavů, matice přechodových pravděpodobností a počátečního rozdělení pravděpodobnosti a funkcemi hustoty rozdělení pravděpodobnosti. Diskrétní HMM (tj. pozorované symboly jsou z konečné množiny symbolů) můžeme zapsat jako uspořádanou trojici $\lambda = (A, B, \Pi)$ [17]:

- N – počet stavů modelu. Je-li S množina stavů, pak $S = \{S_1, S_2, \ldots, S_N\}$. Stav modelu v čase t je dán $q_t \in S, 1 \leq t \leq T$, kde T je délka sledované sekvence.
- M – počet různých sledovaných symbolů. Pokud V je množina sledovaných symbolů, pak $V = \{v_1, v_2, \ldots, v_M\}$.
- A – matice přechodových pravděpodobností, $A = \{a_{ij}\}$, kde:

  $$a_{ij} = P\left[q_t = S_j | q_{t-1} = S_i\right], \quad 1 \leq i, j \leq N,$$

  s tím, že:

  $$0 \leq a_{i,j} \leq 1, \quad \sum_{j=1}^{N} a_{i,j} = 1, \quad 1 \leq i \leq N.$$



Obrázek 4.13: Využití modelu pro identifikaci. Databáze je vytvořena použitím modelu na trénovací snímky. Výsledkem adaptace modelu jsou koeficienty a_i a b_i, které jsou uloženy. Při identifikaci jsou koeficienty získané z nasnímaného obličeje porovnány s databází. Upraveno z [4].

- B – matice pravděpodobností pozorovaných symbolů, $B = \{b_j(k)\}$, kde:

$$b_j(k) = P\left[\mathbf{O_t} = v_k | q_t = S_j\right],$$

$$1 \leq j \leq N, \quad 1 \leq k \leq M$$

  a $\mathbf{O_t}$ je symbol pozorovaný v čase t.

- Π – počáteční rozdělení pravděpodobnosti, $\Pi = \{\pi_i\}$, kde:

$$\pi_i = P\left[q_1 = S_i\right], \quad 1 \leq i \leq N.$$

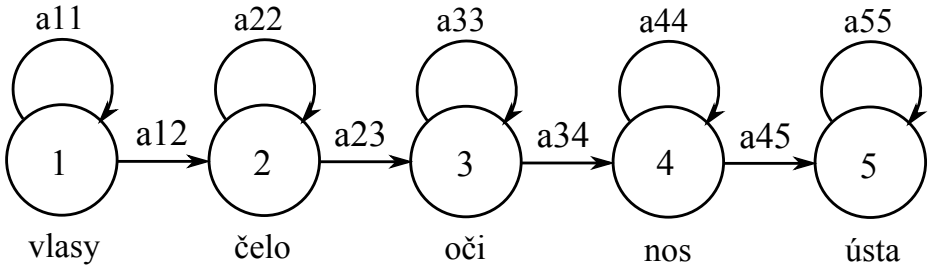
Nejobecnější reprezentací funkce hustoty rozdělení pravděpodobnosti je konečná směs ve tvaru:

$$b_i(\mathbf{O}) = \sum_{k=1}^{M} c_{ik} N\left(\mathbf{O}, \mu_{ik}, U_{ik}\right), \quad 1 \leq i \leq N, \tag{4.33}$$

kde c_{ik} je směšovací koeficient k-té směsi ve stavu i. Bez ztráty na obecnosti můžeme předpokládat, že $N\left(\mathbf{O}, \mu_{ik}, U_{ik}\right)$ je Gaussovo rozdělení pravděpodobnosti se středním vektorem μ_{ik} a kovarianční maticí U_{ik}.

U rozpoznávání řeči, kde se HMM používají, jsou zpracovávaná data jednorozměrná. Rozšíření na dvourozměrné HMM za účelem rozpoznávání obličejů je možné, ale ukázalo se

jako výpočetně náročné. Nicméně je možné použít i jednorozměrné modely a to tím způsobem, že snímek s obličejem rozdělíme na regiony s pro rozpoznávání významnými oblastmi (vlasy, čelo, oči, nos, ústa) a tyto zpracováváme postupně shora dolů. Každému takovému regionu je přiřazen jeden stav jednorozměrného skrytého Markovova modelu (viz. Obr. 4.14).



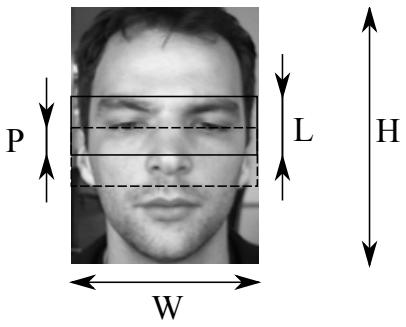
Obrázek 4.14: HMM pro rozpoznávání obličeje. Upraveno z [17].

Extrakce příznaků

Každý snímek výšky H a šířky W je rozdělen do překrývajících se bloků výšky L a šířky W. Velikost překrytí sousedních bloků je P. Z toho vyplývá, že počet bloků a tedy i počet pozorovaných vektorů bude:

$$T = \frac{H - L}{L - P} + 1. \tag{4.34}$$

Volba parametrů L a P může významně ovlivnit úspěšnost rozpoznávání. Velké překrytí bloků úspěšnost značně zvyšuje. Volba parametru L je složitější a při malých nebo velkých hodnotách zvyšuje chybovost. Nicméně, pokud je zvoleno velké P ($P \leq L - 1$), je vliv L na úspěšnost malý [17].



Obrázek 4.15: Rozdělení snímku do regionů.

Pozorovaný vektor se skládá z hodnot všech pixelů v daném bloku. Jeho velikost je tedy $L \times W$. Použití hodnot pixelů má ovšem nevýhody – nereprezentují vhodně vlastnosti bloku, protože jsou ovlivnitelné šumem, posunutím a rotací obrazu, změnou v nasvícení a podobnými vlivy. Dále velké rozměry bloků a tedy vektorů vedou k velké výpočetní náročnosti. Proto je vhodné příznaky z bloků extrahovat jinak, např. použitím 2D diskrétní kosinové transformace (DCT) a koeficienty DCT použít jako vektory popisující bloky [17].

Trénování

Každý jednotlivec uložený v databázi je reprezentován jedním modelem obličeje. Sada různých trénovacích snímků jedné osoby je použita k natrénování jednoho HMM. Snímky jsou rozděleny na bloky, z nichž jsou extrahovány DCT koeficienty, použité pro trénování. Nejprve je inicializován HMM, trénovací data jsou rovnoměrně shora dolů rozdělena do stavů a vektory reprezentující bloku v daných stavech jsou použity pro odhad pravděpodobnostní matice B. Výchozí hodnoty pro A a Π jsou dány strukturou modelu. V dalším kroku jsou parametry modelu přehodnoceny pomocí metody *Expectation-maximization* (algoritmus pro výpočet maximální likelihood – věrohodnosti) za účelem maximalizace $P(\mathbf{O}|\lambda)$. Iterační výpočet se zastaví po dosažení konvergence, tj. když je rozdíl v pravděpodobnostech v dvou po sobě jdoucích krocích menší než zvolený práh C [17].

$$|P\left(\mathbf{O}|\lambda^{(k+1)}\right) - P(\mathbf{O}|\lambda^{(k)}\Big)| < C. \tag{4.35}$$

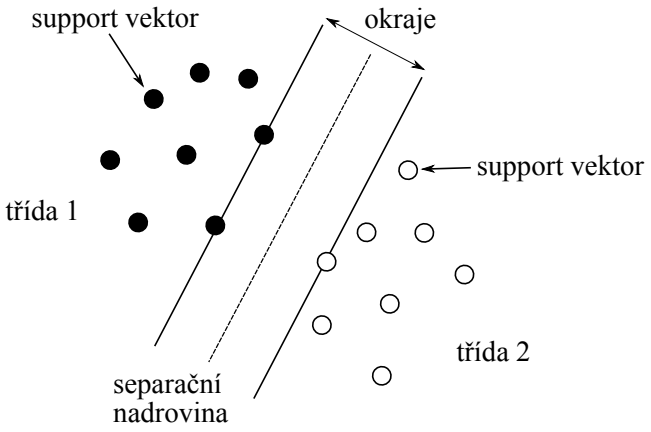
Rozpoznávání

Při rozpoznávání jsou nejprve ze vstupního snímku extrahovány vektory jako v trénovací fázi. Poté je vypočtena pravděpodobnost vektoru pro každý model obličeje. Snímek t je klasifikován jako obličej k v případě, že:

$$P\left(\mathbf{O}^{(t)}|\lambda_k\right) = max_n P\left(\mathbf{O}^{(t)}|\lambda_n\right). \tag{4.36}$$

4.6 Support vector machine (SVM)

Tato metoda provádí klasifikaci dat vytvořením separační N–dimenzionální nadroviny. Tato nadrovina optimálně rozděluje data do dvou kategorií. K výpočtu okraje mezi nimi jsou zkonstruovány dvě paralelní nadroviny, na každé straně separační nadroviny jedna. Čím větší je potom vzdálenost každé z nadrovin od bodů z opačné kategorie, tím menší je chyba klasifikace [12].



Obrázek 4.16: Příklad SVM klasifikace v 2D prostoru.

Ke klasifikaci do více než dvou tříd je třeba zkombinovat více SVM. Např. jsou jednotlivé SVM uspořádány do binárního stromu a klasifikace probíhá postupně po úrovních stromu

odspoda nahoru. Provede se klasifikace v každé větvi podstromu a vítězná třída postupuje výše. Výsledkem je jediná třída, která se dostane až do kořene stromu [12].

Kapitola 5

Implementace

V této kapitole jsou uvedeny postupy při implementaci zvolených algoritmů v jazycích Java a C++ pro platformu Android. Je zde popsána struktura aplikace a podrobně jsou rozebrány důležité části programu.

5.1 Použité technologie

Vývoj aplikací pro Android probíhá v jazyce Java, avšak v některých případech, kdy je třeba optimalizace kvůli náročnějším výpočtům, nebo použití knihoven, je vhodné uchýlit se k nižším jazykům, tj. C nebo C++ a tyto části aplikace vytvořit v nativním kódu. Tak je tomu i v tomto případě. V Javě jsou napsány části zajišťující získání snímku, detekci obličejů na něm a práci s databází (tj. nalezení identit na základě získaných feature vektorů). Rovněž veškeré grafické rozhraní nemělo smysl vytvářet jinak. Blok zajišťující extrakci feature vektorů z snímku konkrétního obličeje je pak vytvořen v jazyce C++ a je připojen ke zbytku aplikace jako sdílená knihovna.

Při zpracování snímků je z výhodou využita knihovna OpenCV (4.1.1), jejíž zprovoznění a propojení s ostatním kódem bylo poměrně velkým oříškem. V době počátku tvorby této práce byl port knihovny pro Android pouze neoficiální. Po čase se stal ovšem přímou součástí projektu a vyšlo několik nových verzí, ovšem i přesto byly obrovské problémy s překladem způsobené obskurními chybami při linkování jednotlivých částí knihovny a projevující se až nevysvětlitelnou nefunkčností aplikace a pády při načítání knihovny.

Pro identifikaci byl zvolen algoritmus AAM (4.4.3), i přesto, že se už zpočátku jevil jako výkonově náročnější. Nicméně použití např. klasické Analýzy hlavních komponent se zdálo jako značně ulehčující práci (algoritmus je již implementován v knihovně OpenCV), tudíž byl zvolen tento postup.

5.2 Struktura aplikace

Celá podoba aplikace byla z počátku vývoje zamýšlena tak, aby mohla spolupracovat s jinými aplikacemi. Bylo tedy nutné zvolit vhodný způsob implementace části programu pro identifikaci, aby mohla pracovat samostatně. V konečné podobě je blok zajišťující rozpoznání osob, práci s databází a klasifikaci provozován jako tzv. *remote service*. Jde tedy o službu, která je primárně spouštěna po startu této aplikace, ale teoreticky je možné ji spustit i externě. Součástí je i klientská část určená víceméně pro testování.

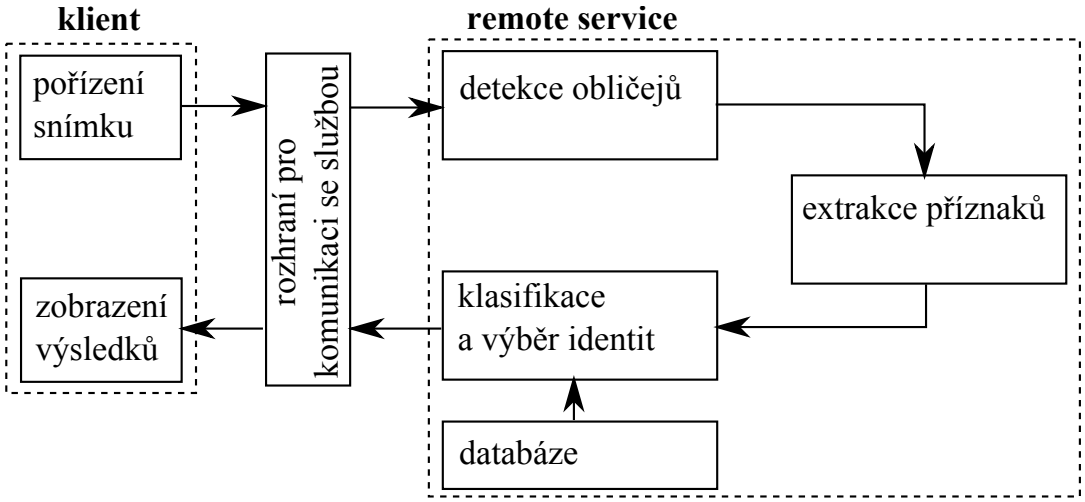
Klientská Aktivita funguje jako galerie, která zobrazuje snímky uložené v externí paměti telefonu (většinou SD karta), konkrétně ve složce `/sdcard/Photo` (což se ovšem může měnit v závislosti na konkrétním přístroji a verzi operačního systému). Pro vybraný snímek je možné spustit identifikaci, která probíhá v samostatném procesu a po jejímž skončení jsou na snímku označeny regiony s nalezenými obličeji. Pro nalezené obličeje je možné stiskem zobrazit podrobnosti – otevře se dialog s odhadnutými identitami, nebo je možné neznámý obličej přidat do databáze.

5.3 Průběh identifikace

Prvním krokem celého procesu je získání vstupního snímku. O toto se stará klientská aplikace, tedy v našem případě Aktivita pracující jako galerie a zobrazující snímky z externí paměti. Původním záměrem bylo vytvořit i další část, která bude získávat obrázky z kamery fotoaparátu, ovšem kvůli časové náročnosti celé identifikace a složitosti ostatních částí se bohužel na podobné funkce nedostalo.

Získaný snímek je předán službě s požadavkem k identifikaci. Nejprve jsou pomocí detektoru, který je součástí Android API, nalezeny všechny obličeje. Pro každý obličej je použita funkce pro extrakci příznaků, kterou poskytuje sdílená knihovna napsaná v nativním kódu. Její funkčnost bude podrobněji popsána dále. Výsledkem tohoto je vektor hodnot (konkrétně je používán vektor reálných čísel délky 16), definující daný obličej.

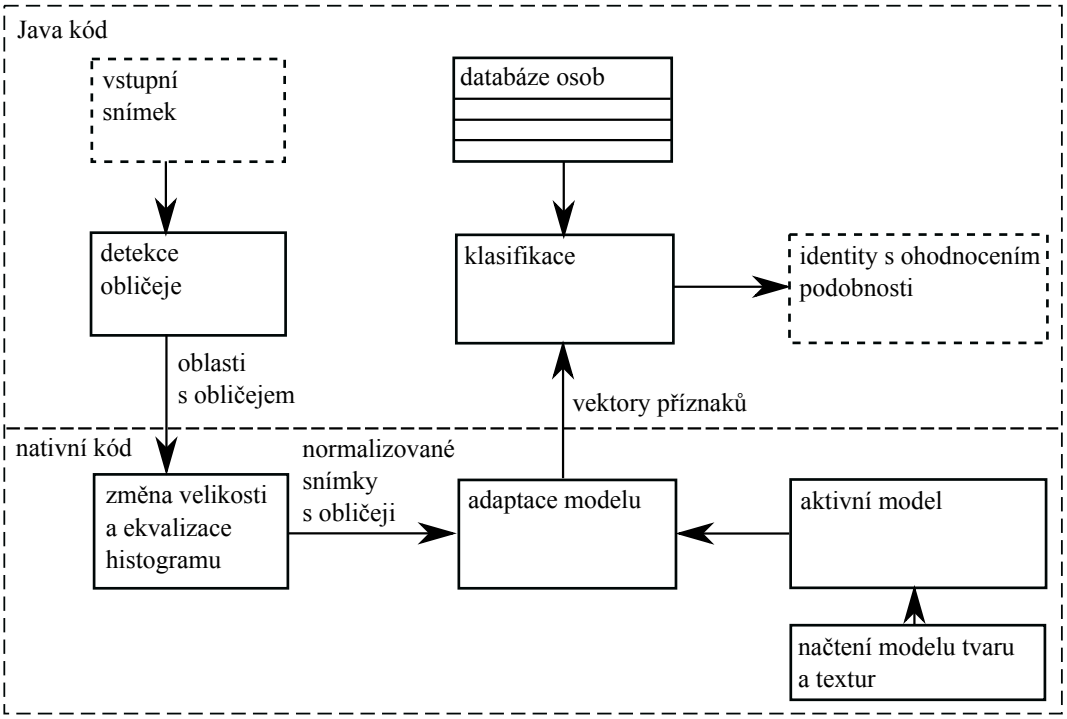
V posledním kroku je kontaktována databáze, která na základě vektoru příznaků a zvoleného nastavení hledání přiřadí každému obličeji několik pravděpodobných identit. U každé identity je uvedena i podobnost popsaná hodnotou v intervalu $\langle 0, 100 \rangle$ a uváděná v procentech, ovšem její hodnota je spíše orientační a k porovnání výsledků mezi sebou.



Obrázek 5.1: Průběh identifikace.

5.4 Popis bloků aplikace

V této části jsou podrobněji popsány jednotlivé fáze identifikace, vyjma implementace aktivního modelu, který je popsán v samostatné sekci.



Obrázek 5.2: Schema celé aplikace.

5.4.1 Klientská část

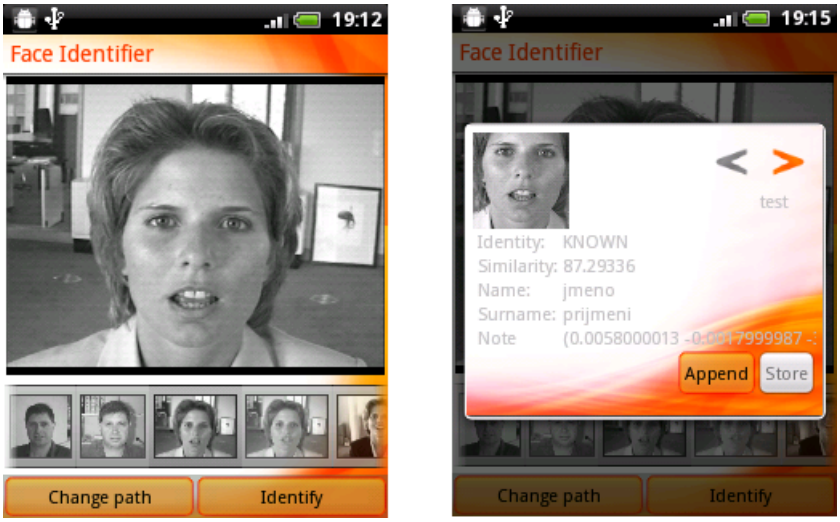
Klientská část aplikace je tvořena z několika Aktivit. První, která se spouští jako hlavní Aktivita aplikace, slouží pouze jako menu pro výběr operací. Hlavní část klienta slouží pro výběr a prohlížení obrázků z externí paměti. Pro vybraný snímek je možné spustit identifikaci tlačítkem *Identify*. Identifikace se spouští pomocí třídy zděděné od třídy `AsyncTask`, takže běží na pozadí a neblokuje další činnost aplikace. Po ukončení výpočtu je jako reakce zavolána metoda `onIdentificationDone()` rozhraní `IdentificationEvent` a Aktivitě je předán seznam rozpoznaných identit.

Rozpoznané obličeje jsou na snímku zvýrazněny červeným obdélníkem. Dotykem je možné zobrazit dialog s informacemi o vybraném obličeji. Pokud byl obličej nalezen v databázi, jsou v dialogu uvedeny informace o osobě. Je-li nalezeno více podobných osob, je možné mezi nimi přepínat. U osob je kromě jména a příjmení zobrazena informace o podobnosti s osobami uloženými v databázi. Vektory nalezených obličejů je možné navíc do databáze přidat. Ať už k existující osobě, nebo osobě nové.

Třída `Identity`

`Identity` je třída reprezentující identitu osoby, obsahuje informace o jménu, příjmení a další, používané pouze v některých situacích a při některých fázích identifikace.

- `id` – unikátní celočíselný identifikátor, je generován při ukládání nové osoby do databáze. Při vzniku objektu je nastaven na zápornou hodnotu.
- `type` – tento parametr se mění v průběhu cesty objektu od nalezení obličeje na snímku, kdy objekt třídy `Identity` vzniká, až po distribuci výsledků a případné uložení rozpo-



Obrázek 5.3: Snímky z aplikace. Galerie (vlevo) a dialog pro prohlížení identity osoby (vpravo).

znaných dat do databáze. Po vytvoření a detekci obličeje, kdy má identita přidělený obrázek nalezeného obličeje, je nastaven na hodnotu `UNSET`. Po extrakci vektoru příznaků je pak změněn na `INDEFINABLE`, pokud se z nějakého důvodu nepodaří příznaky extrahovat, nebo `UNKNOWN`, pokud proběhne v pořádku. Následuje vyhledání podobných osob v databázi. Pokud je nalezena podobná osoba, je typ nastaven na `KNOWN` a současně je nastaven parametr podobnosti. Posledním stavem je `STORED`, který se používá pouze pro informaci, že identita byla právě přidána do databáze.

- `name` – jméno osoby.
- `surname` – příjmení osoby.
- `similarity` – podobnost osoby. Tento parametr je nastavován při klasifikaci a slouží k přibližnému porovnání podobnosti osoby nalezené na vstupním snímku a osob uložených v databázi. Jeho hodnota je v intervalu $\langle 0,\ 100 \rangle$ a je uváděn v procentech.
- `faceImage` – snímek obličeje nalezený na vstupním snímku.
- `rectangle` – oblast na vstupním snímku, kde se obličej nachází.
- `vector` – vektor příznaků extrahovaný ze snímku obličeje.

Tato třída implementuje rozhraní `Comparable<Identity>`, aby bylo možné identity mezi sebou porovnávat. Porovnávání probíhá pouze porovnáním hodnot atributu `id`. Při použití v databázi je tato třída rozšířena na `RatedIdentity`, aby bylo možné ji porovnávat podle jiných kritérií. Dalším implementovaným rozhraním je `Parcelable`. *Parcelace* je podobná serializaci, Identita je zapsána do objektu třídy `Parcel`, který je možné posílat při komunikaci mezi klientskou aplikací a službou.

5.4.2 Detekce obličeje

V původním plánu bylo, aby detektor obličeje byl součástí nativního kódu. K detekci měly posloužit funkce poskytované knihovnou OpenCV, tzn. detekce pomocí Haarova klasifikátoru. Implementace se podařila, ovšem po otestování vyšlo najevo, že detekce z nějakého důvodu probíhá neúměrně dlouho (řádově desítky vteřin) i na relativně malých obrázcích (cca 200 × 200 pixelů). Z tohoto důvodu bylo přistoupeno k použití detektoru, který poskytuje přímou API systému Android.

K detekci obličejů na platformě Android slouží třída `FaceDetector`, jejíž instanci je potřeba vytvářet pro každý nový snímek. Jako parametry přijímá rozměry snímku a maximální počet obličejů, který má být detekován. Tento počet byl při testování pracovně nastaven na hodnotu 10. Detekce je spuštěna voláním metody `findFaces()` s dvěma parametry: vstupním snímkem a objektem pole s již připravenou velikostí pro uložení výsledných objektů typu `FaceDetector.Face`.

Nalezené obličeje jsou definované polohou jejich středu a vzdáleností očí. Oblast obličeje je tedy nutné z těchto hodnot nějak vhodně spočítat. Po krátkém testování byl zvolen následující výpočet oblasti s obličejem pomocí vhodných konstant:

```
region.left   = (int) (midPoint.x - 1.25 * eyeDistance);
region.right  = (int) (midPoint.x + 1.25 * eyeDistance);
region.top    = (int) (midPoint.y - 0.6  * eyeDistance);
region.bottom = (int) (midPoint.y + 1.9  * eyeDistance);
```

Kde `region` je obdélník (resp. v tomto případě čtverec) ohraničující obličej, `midPoint` je střed obličeje a `eyeDistance` je vzdálenost očí.

Stejné nastavení detektoru bylo mimojiné použito i u označování oblastí pro snímky určené k trénování identifikátoru.

Výsledkem detekce je vytvoření objektu `FIResults`, obsahujícího pro každý nalezený obličej nový objekt třídy `Identity`, který má přiřazenou bitmapu s výřezem vstupního snímku a je připraven k extrakci příznaků.

5.4.3 Extrakce příznaků

Cílem extrakce příznaků je získat pro každý snímek obličeje vektor hodnot, který by ho vhodně popisoval. Za tímto účelem je implementován algoritmus AAM formou sdílené knihovny v nativním kódu. Podrobný popis tohoto algoritmu je v sekci 5.5.

Rozhraní pro komunikaci s knihovnou

Jelikož mezi částmi kódu v Javě a v C++ je potřeba předávat některé objekty (knihovně jsou předávány bitmapy s obličeji a ta následně vrací vektor reálných čísel), bylo pro tento účel potřeba vytvořit rozhraní. Prvním problémem je inicializace knihovny. Po jejím připojení je třeba načíst data pro aktivní model, tj. načíst soubor s informacemi o tvaru a soubory s texturami a dále vytvořit všechny potřebné objekty. K tomuto účelu je vhodné použít funkci `JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM *, void *)`, která se volá automaticky při načtení knihovny.

Hlavičkový soubor pro rozhraní je možné po definování prototypů metod s klíčovými slovy `native` vygenerovat utilitou `javah` s parametry `-jni` a názvem třídy včetně balíčku (ve tvaru `"balicek1.balicek2.Trida"`). Javovské objekty předané jako parametry jsou proměnné typu `jobject`.

Pro převedení Bitmapy z objektu v Javě do objektu `Mat` knihovny C++ bylo třeba zjistit strukturu dat v této třídě a vytvořit vlastní funkci, která data překonvertuje. Implementovaná funkce pracuje pouze s Bitmapami ve formátu `ANDROID_BITMAP_FORMAT_RGB_565`, kde je každý pixel uložen na dvou bytech s tím, že prvních pět bitů patří červené složce, dalších šest zelené a zbylých pět modré složce modelu RGB. Při konverzi je snímek rovnou převeden na odstíny šedi, jelikož vytvořená knihovna pracuje pouze s šedotónovými snímky.

5.4.4 Databáze a klasifikace

Po zjištění vektorů zbývá nalezení nejbližších hodnot mezi vektory uloženými v databázi. Pro výběr nejbližších identit je připraveno několik módů pro třídu zodpovědnou za komunikaci s databází (`IdentitySelector`) – výběr jednoho nejbližšího záznamu (`ONE_NEAREST`), výběr n nejbližších záznamů (`N_NEAREST`), výběr všech záznamů do zvolené vzdálenosti (`UP_TO_DISTANCE`) a výběr n záznamů do zvolené vzdálenosti (`N_NEAREST_UP_TO_DISTANCE`).

Pro testovací databázi byla zvolena jednoduché ukládání záznamů do souboru XML. Nicméně součástí je rozhraní pro práci s databází (`FIDatabase`), které při jeho dodržení umožňuje vytvořit jinou implementaci bez jakéhokoliv zásahu do ostatního kódu. Rozhraní obsahuje prototypy metod pro připojení a odpojení od databáze a pro čtení a vkládání záznamů.

Současná implementace uchovává data v XML souboru uloženém v externí paměti. Při připojení k databázi je soubor otevřen a rozparsován. Pokud se soubor nepodaří otevřít, je vytvořen nový se základní strukturou. Stav databáze je ukládán při každé změně.

Ke klasifikaci, tj. k výběru třídy, která je nejbližší získanému vektoru, je použit jednoduchý klasifikátor nejbližší vzdálenosti. Každá třída je reprezentována jedním vektorem, který je průměrem všech již uložených vektorů, náležících této třídě. Ke vzdálenosti vektoru třídy od nově získaného vektoru je použita Euklidovská metrika.

5.5 Implementace AAM

Tato část popisuje implementaci algoritmu AAM, který byl vybrán pro extrakci vektoru příznaků ze vstupního snímku.

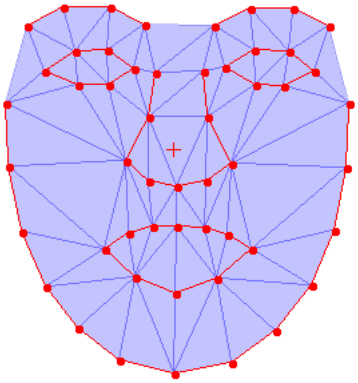
5.5.1 Struktura modelu

Před implementací bylo třeba zvolit vzhled a strukturu modelu. K popisu je nakonec použito celkem 53 bodů – 13 z nich ohraničuje obličej podél jeho spodní části až zhruba do výšky uší, 10 bodů je umístěno na hranici úst, 9 popisuje tvar nosu, 6 bodů je určeno pro každé oko a 4 další pro horní část obočí. Propojení do trojúhelníků bylo zvoleno tak, aby při deformacích tvaru pokud možno nedocházelo k jejich převracení a překrývání.

5.5.2 Získání trénovacích dat

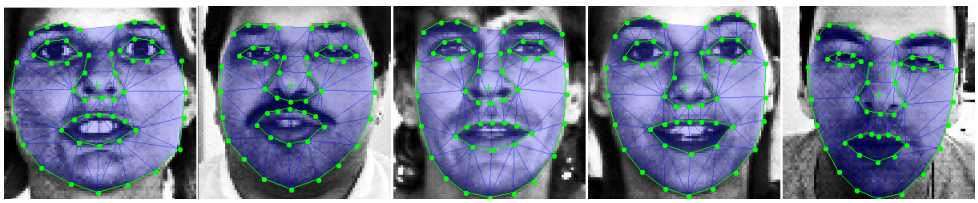
Pro natrénování modelu je třeba nejprve získat vhodné snímky s obličeji. Snímky byly vybrány z databáze *BioID-FaceDatabase-V1.2*. Prvním krokem byla detekce obličejů detektorem, který poskytuje API Androidu. Ořezané snímky byly zmenšeny na velikost 80×80 pixelů a byla na nich provedena ekvalizace histogramu.

Následuje část, kterou není možné jednoduše zautomatizovat – je nutné vyznačit na všech snímcích místa, kde se vyskytují body tvaru. Pro tento účel byla vytvořena jednoduchá jednoúčelová desktopová aplikace, která umožňuje zobrazování trénovacích snímků



Obrázek 5.4: Struktura implementovaného modelu tvaru.

a vyznačování bodů pomocí myši. Program neoplývá zvlášť propracovaným grafickým rozhraním, ale je schopný načítat a ukládat modely zapsané do XML souboru a exportovat je do textového formátu, který využívá již samotná mobilní aplikace. Počáteční strukturu souboru bylo ovšem nutné vytvořit také ručně. Vytváření jiných trénovacích sad je tedy vhodné provádět úpravou již existujících souborů a tento nástroj pro anotaci použít pouze pro změnu polohy bodů.



Obrázek 5.5: Ukázka anotovaných obličejů.

Ve finální podobě je model trénován 64 vzorky, to znamená přemístění více jak tří tisíc bodů, takže tato práce je poměrně náročná (celkově zabrala anotace nejméně dvě hodiny). Je vhodné, aby označování tvarů prováděla jedna osoba a pokud možno, aby byly všechny vzorky označeny bez časových prodlev, aby byla zachována korespondence bodů napříč všemi vzorky.

Ve výsledku je model, tak jak je používán pro trénování na mobilním zařízení, tvořen textovým souborem s popisem tvaru a seznamem textur a soubory obrázků s texturami. Tato data jsou umístěna v balíčku s aplikací v adresáři `assets`. Odtud jsou při prvním spuštění zkopírována do interní paměti telefonu, která je přidělena aplikaci a odkud je jednodušší je číst z nativního kódu. Formát textového souboru je poměrně jednoduchý, aby bylo snazší jeho zpracování – na začátku jsou informace o množství vzorků a počtu bodů tvaru, následují řádky uvozené slovem `LINE` a `TRIANGLE` následovaném indexy bodů pro informaci o propojení bodů. Na řádcích začínajících slovem `SHAPE` jsou uložené souřadnice všech bodů jednoho tvaru a po slově `TEXTURE` je uvedeno jméno souboru s texturou, přičemž záleží na pořadí uvedených dat.

5.5.3 Trénování modelu

Celý model je rozdělen na dvě samostatné části – model tvaru (třída `ShapeModel`) a model textur (třída `TextureModel`). Natrénování tvaru je jednoduché, je vytvořen objekt popisující strukturu, tj. propojení bodů do trojúhelníků, a pro každý trénovací vzorek je uložen seznam souřadnic. Jelikož počet vzorků a bodů tvaru může být vysoký a tím pádem by byl velký i vektoru popisující tvar, je pro snížení dimenze použito PCA. Souřadnice všech bodů jsou uloženy do matice tak, že každý řádek reprezentuje jeden vzorek. Na řádku jsou zapsané všechny souřadnice tak, že nejprve jsou uloženy všechny x-ové souřadnice a za nimi všechny y-ové. Vznikne tak matice o rozměrech $2M \times N$, kde M je počet bodů tvaru a N je počet trénovacích vzorků:

$$M_s = \begin{pmatrix} x_1^1 & \dots & x_m^1 & y_1^1 & \dots & y_m^1 \\ x_1^2 & \dots & x_m^2 & y_1^2 & \dots & y_m^2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1^n & \dots & x_m^n & y_1^n & \dots & y_m^n \end{pmatrix},$$

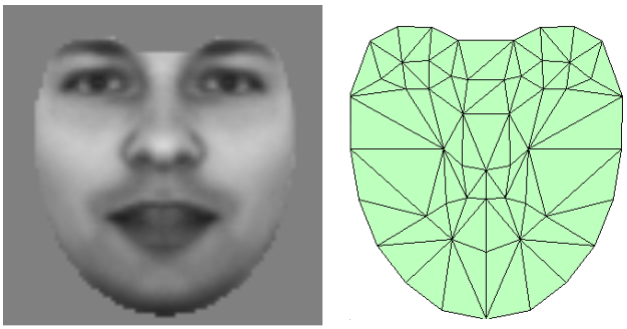
kde x_i^j a y_i^j pro $0 < i < m$ a $0 < j < n$ jsou souřadnice i-tého bodu j-tého vzorku. V tomto okamžiku je s výhodou využita knihovna OpenCV, která poskytuje třídu `PCA` pro výpočet analýzy hlavních komponent. Konstruktor této třídy přijímá 4 argumenty – matici s daty (tj. matice M_s), jednořádkovou matici pro uložení průměrného vektoru, informaci, zda jsou vzorky uloženy v řádcích nebo sloupcích, a maximální počet komponent. Poslední parametr bude udávat velikost vektoru, kterým bude popsán vygenerovaný tvar. Pro tento i pro vektor pro textury byla zvolena velikost 8. Výsledný vektor popisující vygenerovaný obličej vznikne konkatenací těchto dvou a bude mít tedy velikost 16.

Použití vzniklého objektu `PCA` je jednoduché, knihovna se stará o všechny potřebné výpočty. Metodami `project()` resp. `backProject()` je možné promítnout vektor tvaru do redukovaného prostoru a naopak, z vektoru v redukovaném prostoru vytvořit nový vektor tvaru.

Příprava modelu textury je o něco náročnější. Pro jeho vytvoření je třeba mít nachystaný model tvaru, resp. mít souřadnice bodů průměrného tvaru obličeje. Každý vzorek textury je načten do matice a je mu přiřazen jeho příslušný tvar. Poté jsou použitím warpingu všechny vzorky přetransformovány z jejich původního tvaru do průměrného, tím dojde vlastně k zarovnání všech trénovacích dat. Takto upravené textury jsou stejně jako v předchozím případě uloženy do jedné matice a to tak, že na každém řádku jsou informace o pixelech jednoho vzorku. Velikost matice je opět velká pro přímé použití – N řádků a $(W \times H)$ sloupců, kde W je šířka obrázku s texturou H je výška. Stejně jako u tvaru vhodné použít PCA analýzu a opět je zvolen maximální počet komponent osm. Je tedy možné z vektoru osmi hodnot vytvořit texturu obličeje s průměrným tvarem.

Warping textur

Funkce pro warping je poměrně důležitou součástí a je třeba jí věnovat pozornost. Zvláště protože je velice často volána a její časová náročnost má velký vliv na dobu celého procesu identifikace. Původně byla implementována kompletně vlastní verze, která využívala barycentrických souřadnic. Výpočet nové textury probíhal tak, že byla vytvořena matice pro výslednou bitmapu. Procházením pixel po pixelu se zjišťovalo, zda daný bod leží uvnitř některého z trojúhelníků nového tvaru. Pokud ano, byly vypočteny barycentrické souřadnice bodu v tomto trojúhelníku, nalezen korespondující trojúhelník ve starém tvaru a vypočteny



Obrázek 5.6: Průměrný obličej (textura vlevo a samostatný tvar vpravo), tj. obličej vygenerovaný z vektoru, jehož všechny složky (jak pro tvar, tak pro texturu) jsou nulové.

kartézské souřadnice ve výchozí textuře. Pro výpočet intenzity pixelu nebyla ani použita interpolace, pouze se použila hodnota nejbližšího pixelu. Tato funkce se ovšem při průběžném testování ukázalo jako časově velmi náročná a tak byla hledána jiná možnost.

Funkci pro afinní warping poskytuje i knihovna OpenCV. Problémem ale je, že se warping provádí nad celým obrázkem. Transformace jednoho trojúhelníku tedy probíhá tak, že je nejprve vytvořen obrázek s jeho binární maskou. Pomocí funkce `getAffineTransform()`, která dostane jako parametry souřadnice bodů výchozího a nového trojúhelníku, je získána transformační matice. Aby se transformace neprováděla nad všemi pixely textury, je použita možnost nastavit tzv. *Region of interest*, tj. obdélníková oblast okolo trojúhelníka, pro kterou se má operace provést. Na tuto oblast je použita transformační matice. Z takto vzniklé bitmapy jsou podle nastavené masky zkopírovány pixely uvnitř trojúhelníku do cílové textury. Tento postup se ukázal být asi desetkrát rychlejší než předešlý.

5.5.4 Adaptace modelu

Cílem procesu adaptace je vygenerování takového obličeje, který se co nejvíc podobá vstupnímu snímku. Vzdálenost zadaného a vygenerovaného snímku je počítána funkcí `norm()`. U ní je možné zvolit, jaká metrika má být pro výpočet vzdálenosti použita. V našem případě je použit parametr `NORM_L1` a výpočet vypadá následovně: $d = \sum_I |img_1(I) - img_2(I)|$. Jde vlastně o Manhattanskou vzdálenost (3.3).

Výsledný obličej je generován v iteračním cyklu, ve kterém jsou odhadovány a zpřesňovány hodnoty prvků vektorů $\mathbf{s} = (s_1, \ldots, s_8)$ pro popis tvaru a $\mathbf{t} = (t_1, \ldots, t_8)$ pro popis textury.

V každém kroku iterace jsou nejprve provedeny afinní transformace (tj. rotace, posun a změna měřítka) tvaru tak, aby se minimalizovala chyba mezi vstupním a generovaným snímkem. Dále je prováděna změna tvaru – ke prvnímu prvku vektoru s je přičtena aktuální hodnota kroku (velikosti kroků pro změnu tvaru byly zvoleny ± 0.02, ± 0.002, ± 0.0002). Pokud změna sníží vzdálenost od vstupního snímku, zůstává zachována, jinak je zmenšena velikost kroku a postupuje se dále. Po provedení nejmenšího kroku se pokračuje další komponentou vektoru.

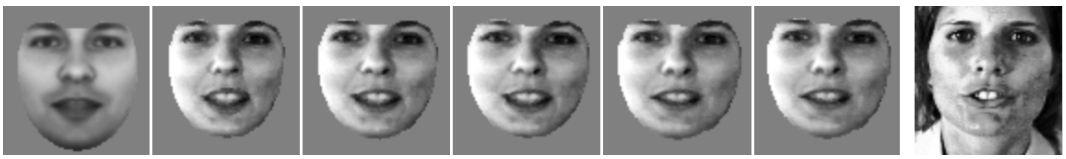
Po každé změně je třeba vygenerovat aktuální vzhled obličeje. Vektor $\mathbf{s}$ je předán modelu tvaru, který zpětnou projekcí z redukovaného prostoru vytvoří nový tvar. Poté je na tento tvar transformována aktuální textura a je možné spočítat rozdíl proti vstupnímu obličeji.

Po tom, co je dosaženo optimálního tvaru, je přistoupeno ke změně textury. Opět jsou

po zvolených krocích upravovány prvky vektoru $\mathbf{t}$, dokud dochází ke zlepšení výsledku. Po každé změně některé z hodnot je vektor předán modelu pro generování textury a ten vytvoří novou texturu pro průměrný tvar. Takovou texturu je ještě třeba warpingem transformovat na aktuální tvar.

Po adaptaci textury pokračuje proces zase afinními transformacemi. Všechny kroky se opakují, dokud dochází ke zlepšení generovaného výsledku, nebo dokud není překročen maximální počet iterací.

Nakonec je z vektorů tvaru a textury vytvořen jejich konkatenací vektor popisující celý obličej, který je výsledkem celého procesu.



Obrázek 5.7: Adaptace modelu. Zleva po jedné až šesti iteracích. Úplně vpravo je vstupní snímek.

Kapitola 6

Výsledky

V této kapitole je popsáno testování implementované aplikace. Jsou uvedeny výsledku testování jak samostatné knihovny s modelem AAM, tak aplikace jako celku. V úvodu jsou zmíněny zdroje použitých trénovacích a testovacích dat.

6.1 Trénovací a testovací data

K natrénování a testování modelu byly použity dvě volně dostupné databáze. První z nich je možné získat na http://www.vision.caltech.edu/html-files/archive.html, obsahuje 450 snímků přibližně 27 unikátních osob. Druhou je databáze BioID-FaceDatabase (http://support.bioid.com/downloads/facedb/index.php) s 1521 snímky 23 osob.

6.2 Testování AAM

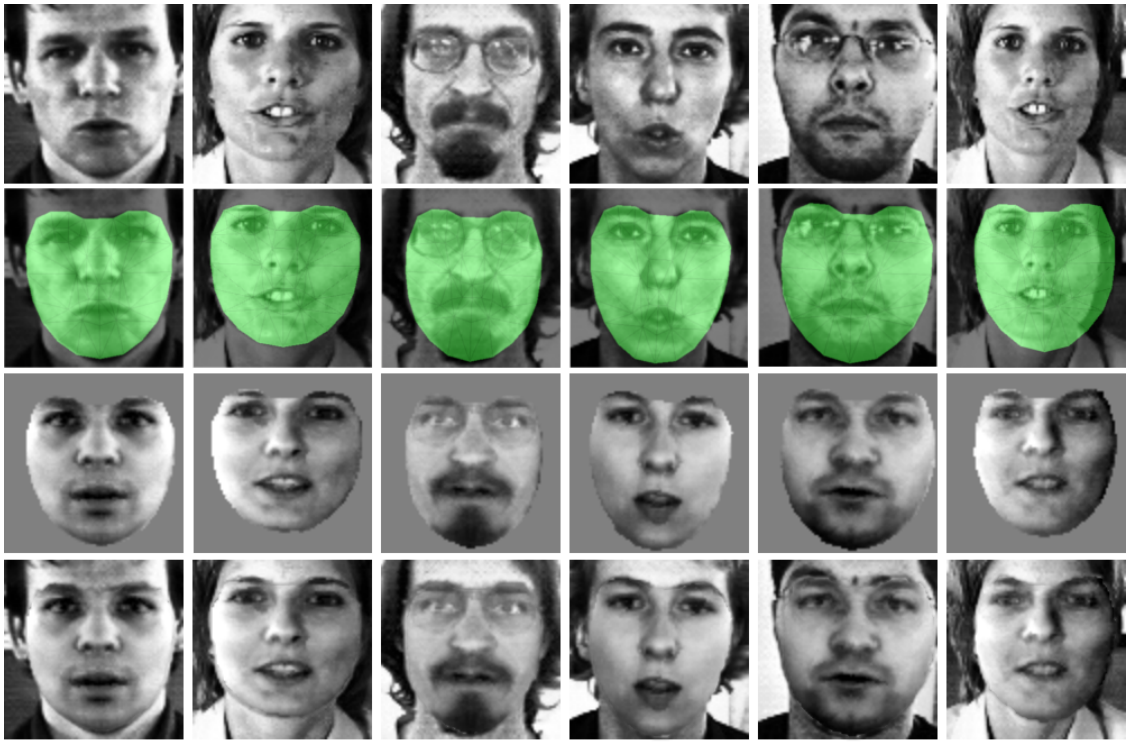
Jádrem celé aplikace je sdílená knihovna pro AAM napsaná v jazyce C++. Jelikož tato část byla vyvíjena samostatně, byla i zvlášť testována. Vstupem této knihovny je snímek s obličejem, kterému má být model přizpůsoben, a výstupem je vektor, který má tento obličej popisovat. K porovnávání výstupních vektorů by bylo třeba vytvořit další aplikaci, což by bylo zdlouhavé a navíc je toto součástí kódu v Javě. Testování tedy probíhalo pouze vizuálním porovnáním vstupních a vygenerovaných obličejů.

Na obrázku 6.1 jsou vidět některé výsledky syntézy obličejů. V horním řádku jsou vstupní snímky tak, jak vypadají po detekci a normalizaci. Rozlišení všech snímků je 80×80 pixelů. Je na ně použita ekvalizace histogramu. Na dalším řádku je možné porovnat generované tvary se vstupem. Poslední dva řádky ukazují výsledné obličeje bez pozadí a s pozadím vstupního obrázku.

Na ukázkách je vidět i několik chyb. Místy plně nekoresponduje vygenerovaný tvar s tvarem vstupního obličeje. U předposledního vzorku je vidět nevhodná adaptace textury, což je pozorovatelné zvlášť v oblasti nosu. Poslední případ ukazuje úplné selhání, které mohlo být způsobenou jak nepřesnou detekcí (výřez s obličejem obsahuje i velkou část krku), tak samotnou adaptací.

Doba výpočtu

Dalším sledovatelným parametrem algoritmu je doba výpočtu vektorů. Konfigurace desktopového počítače, na kterém probíhalo testování je 1400 MB operační paměti, procesor



Obrázek 6.1: Několik ukázek vygenerovaných obličejů. Shora: vstupní snímek, korespondence s vytvořeným tvarem a vytvořený obličej bez pozadí a s pozadím ze vstupu.

s frekvencí 2000 MHz a operační systém Kubuntu 10.10. Mobilním zařízením potom byl telefon HTC Wildfire (frekvence procesoru 528 MHz, 512 MB paměti ROM, 284 MB paměti RAM, operační systém Android 2.2). V tabulce níže jsou uvedeny naměřené časy. Hodnoty udávají pouze dobu adaptace modelu na vstup (i v případě testování na mobilním zařízení), není do nich tedy zahrnuta doba detekce, klasifikace a komunikace se službou. Délka zpracování těchto kroků je ovšem oproti adaptaci zanedbatelná.

PC (s)	39,5	40,0	41,7	39,6	40,1	42,2	40,4	38,6	37,5	36,5	**39,6***
Mobil (s)	0,88	0,89	0,92	0,88	0,91	0,97	0,89	0,88	0,85	0,80	**0,89***

Z testů je vidět, že časy zpracování jsou hodně vysoké (průměrné časy jsou na konci tabulky označené hvězdičkou). Na desktopu tedy adaptace trvá necelou vteřinu, což přibližně odpovídá času, který uvádí autoři metody. Na mobilním zařízení je ale doba zpracování zhruba 40 s, což je prakticky nepoužitelné. Slabým místem algoritmu je warping, který je náročný a je často volán. Jeho optimalizací by se dal čas nejspíš zkrátit. Dalším problémem může být iterační cyklus. Pro použití by ale bylo potřeba zrychlit výpočet alespoň o řád.

6.3 Testování rozpoznávání

Kvůli délce rozpoznávání byla úspěšnost identifikace otestována pouze omezeně. Testování proběhlo na 25 různých osobách a výsledku jsou uvedeny v tabulce níže. V prvním sloupci je ID osoby na vstupním snímku, v dalších je ID prvních tří osob, které systém vybral. Pro každou osobu byly použity čtyři trénovací snímky.

Identita	1. místo	2. místo	3. místo
1	1	24	7
2	2	4	6
3	13	5	7
4	4	18	14
5	5	7	11
6	2	6	18
7	7	11	5
8	8	20	5
9	6	4	8
10	7	10	17
11	7	15	11
12	2	16	1
13	14	2	13
14	14	18	16
15	15	7	6
16	14	6	9
17	17	15	11
18	18	8	20
19	19	15	1
20	5	13	20
21	21	8	3
22	22	20	7
23	23	4	8
24	24	7	11
25	25	17	20

Z 25 osob bylo na první pokus rozpoznáno 16 (úspěšnost cca 64%). Mezi prvními dvěmi rozpoznanými se správná identita vyskytovala v 18 případech (72%) a v 20 případech mezi prvními třemi (80%). Takto krátký test nemá příliš velkou vypovídající hodnotu, ale při době zpracování 40 s na osobu je testování časově náročné.

Úspěšnost není moc vysoká. Ke zlepšení by mohlo přispět delší testování AAM a lepší výběr velikosti iteračních kroků. I samotný cyklus adaptace by určitě bylo možné optimalizovat a tím zvýšit jak úspěšnost, tak rychlost zpracování. Mimo extrakci příznaků ovlivňuje výsledky také klasifikace. Momentálně implementovaný klasifikátor minimální vzdálenosti je možné nahradit sofistikovanějším.

Kapitola 7

Závěr

Cílem této práce bylo seznámit se s platformou Android a navrhnout aplikaci pro identifikaci obličeje, kterou bude možné na této platformě implementovat. Pro implementaci byla vybrána metoda aktivního modelu vzhledu (AAM). Důvodem výběru byla snaha o otestování výkonu současných mobilních zařízení náročnějšími výpočty a vyzkoušení některé netriviální metody pro rozpoznávání. Přes nesnáze se zprovozněním knihovny OpenCV se tato stala důležitým nástrojem a značně ulehčila práci. Byla použita pro práci s obrázky a analýzu hlavních komponent. Díky funkci pro warping, který knihovna poskytuje, se i podařilo urychlit výpočet vektoru příznaků ze snímku obličeje.

Struktura celé aplikace pro identifikaci byla navržena tak, aby mohla být využívána jinými aplikacemi. Vlastní identifikace funguje jako služba, kterou je možné spustit samostatně a která běží na pozadí, dokud není ukončena. Pro komunikaci se službou je vytvořeno rozhraní, přes které je možné volat funkce pro identifikaci, či pro práci s databází. Problémem je ovšem konkurenčnost, se kterou není počítáno. Pokud se tedy ke službě připojí současně více klientů, skončí běh chybou.

Součástí je i klientská část, která nastiňuje použití a propojení služby s aplikací a která slouží víceméně jen pro testování. Jde o jednoduchou galerii, která umožňuje spustit identifikaci pro vybrané snímky, uložené v paměti zařízení.

Pokud jde o praktické využití, byla by aplikace použitelná například pro automatické označování osob na fotkách, uložených v telefonu, nebo přímo při focení. Zajímavá je myšlenka propojení se sociálními sítěmi.

Výsledky implementace ukazují, že zvolená metoda zřejmě není vhodná pro tento účel. Celá aplikace se ukázala jako komplexnější a složitější, než se jevila na počátku. Kvůli tomu nebylo možné věnovat se podrobnostem a důkladnějšímu odladění. Délka výpočtu vektorů je neúnosně dlouhá a ač testování probíhalo na modelu nižší třídy s poměrně nízkým výkonem, lepší zařízení by zřejmě dobu identifikace příliš nezkrátilo. Řešením by mohla být optimalizace algoritmů pro konkrétní procesory. I tak by ale bylo vhodnější zvolit některou méně náročnou metodu, např. PCA.

Úspěšnost rozpoznávání také není vysoká. Příčinou může být pouze částečné zarovnání testovacích dat, nebo nevhodná implementace iteračního cyklu. Pomoci by mohlo i delší testování za účelem lepší volby velikosti iteračních kroků. Počet trénovacích vzorků je zřejmě dostatečný.

Celkově je aplikace spíše experimentální a pro praktické využití nevhodná. Ovšem minimálně celková struktura se ukázala jako vhodně zvolená a po vyřešení některých problémů by mohla být použitelná v praxi.

Literatura

[1] Android Developers: The Developer's Guide [Online]. 2010-12-22 [cit. 2010-12-28]. URL http://developer.android.com/guide

[2] Android Developers: Reference [Online]. 2010-12-22 [cit. 2010-12-28]. URL http://developer.android.com/reference

[3] Blanz, V.; Vetter, T.: A morphable model for the synthesis of 3D faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, ISBN 0201485605, s. 187–194.

[4] Blanz, V.; Vetter, T.: Face recognition based on fitting a 3D morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 25, č. 9, 2003.

[5] Bradski, G.; Kaehler, A.: *Learning OpenCV*. O´Reilly Media, 2008, ISBN 978-0-596-51613-0.

[6] Cootes, T.: Tim Cootes' Home Page. 2011. URL http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/

[7] Cootes, T. F.; Edwards, G. J.; Taylor, C.: Active Appearance Models. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Springer, 1998, s. 484–498.

[8] Cootes, T. F.; Edwards, G. J.; Taylor, C.: Comparing active shape models with active appearance models. In *Proc. British Machine Vision Conf*, BMVA Press, 1999, s. 173–182.

[9] Cootes, T. F.; Taylor, C. J.: Statistical Models of Appearance for Computer Vision. 2004.

[10] Drahanský, M.: Biometrické systémy BIO (Studijní opora k předmětu).

[11] Draper, B. A.; Baek, K.; Bartlett, M. S.; aj.: Recognizing Faces with PCA and ICA. [cit. 2011-4-1]. URL http://www.face-rec.org/algorithms/Comparisons/draper_cviu.pdf

[12] Guo, G.; Li, S. Z.; Chan, K.: Face Recognition by Support Vector Machines. 2000, s. 196–201.

[13] Hinner, J.: Detekce a rozpoznávání obličejů osob a jejich identifikační význam. *Kriminalistika*, 2003.

[14] Kyungnam, K.: Face Recognition using Principle Component Analysis.

[15] Mansfield, A. J.; Wayman, J. L.: Best Practices in Testing and Reporting Performance of Biometric Devices. Technická zpráva, National Physical Laboratory & San Jose State University, August 2002, iSSN 1471–0005.
URL http://www.npl.co.uk/upload/pdf/biometrics_bestprac_v2_1.pdf

[16] Matthews, I.; Baker, S.: Active Appearance Models Revisited. *International Journal of Computer Vision*, ročník 60, 2003: s. 135–164.

[17] Nefian, A. V.; Hayes III, M. H.: Hidden Markov Models For Face Recognition. In *Proc. International Conf. on Acoustics, Speech and Signal Processing*, s. 2721–2724.

[18] Šonka, M.; Hlaváč, V.; Boyle, R.: *Image processing, Analysis and Machine Vision*. Thomson, třetí vydání, 2008, ISBN 978-0-495-24438-7.

[19] Shlens, J.: A tutorial on principal components analysis. In *Systems Neurobiology Laboratory, Salk Institute for Biological Studies*, 2005.

[20] Wikipedia: Dalvik (software) — Wikipedia, The Free Encyclopedia [Online]. 2011-1-2 [cit. 2011-1-3].
URL http://en.wikipedia.org/wiki/Dalvik_(software)

[21] Wikipedia: Mahalanobis distance — Wikipedia, The Free Encyclopedia [Online]. 2011-4-2 [cit. 2011-4-23].
URL http://en.wikipedia.org/wiki/Mahalanobis_distance

[22] Wikipedia: Minkowski distance — Wikipedia, The Free Encyclopedia [Online]. 2011-4-2 [cit. 2011-4-23].
URL http://en.wikipedia.org/wiki/Minkowski_distance

[23] Wikipedia: Taxicab geometry — Wikipedia, The Free Encyclopedia [Online]. 2011-4-2 [cit. 2011-4-23].
URL http://en.wikipedia.org/wiki/Taxicab_geometry

[24] Xiaoguang, L.: Image Analysis for Face Recognition. Dept. of Computer Science & Engineering Michigan State University, East Lansing.

[25] Zhao, W.; Chellappa, R.; Phillips, P.: Subspace Linear Discriminant Analysis for Face Recognition. Technická zpráva, Center of Automation Research University of Maryland & National Institute of Standards and Technology Gaithersburg, 1999.

Příloha A

Obsah CD

- Text technické zprávy ve formátu PDF
- Zdrojové soubory technické zprávy
- Zdrojové kódy aplikace
- Použité databáze snímků obličejů